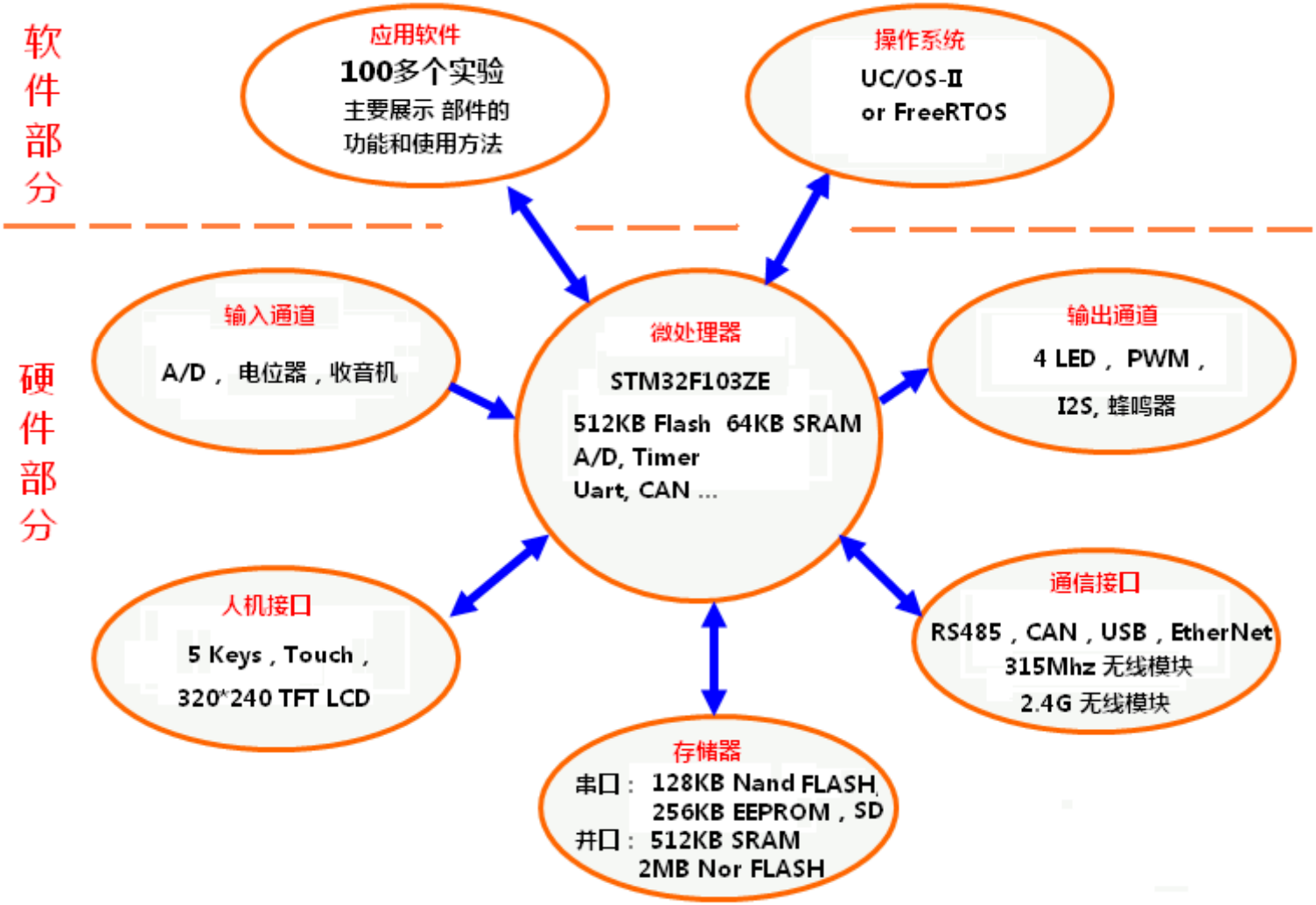
第十四讲 ARM嵌入式系统

结构设计开发实例

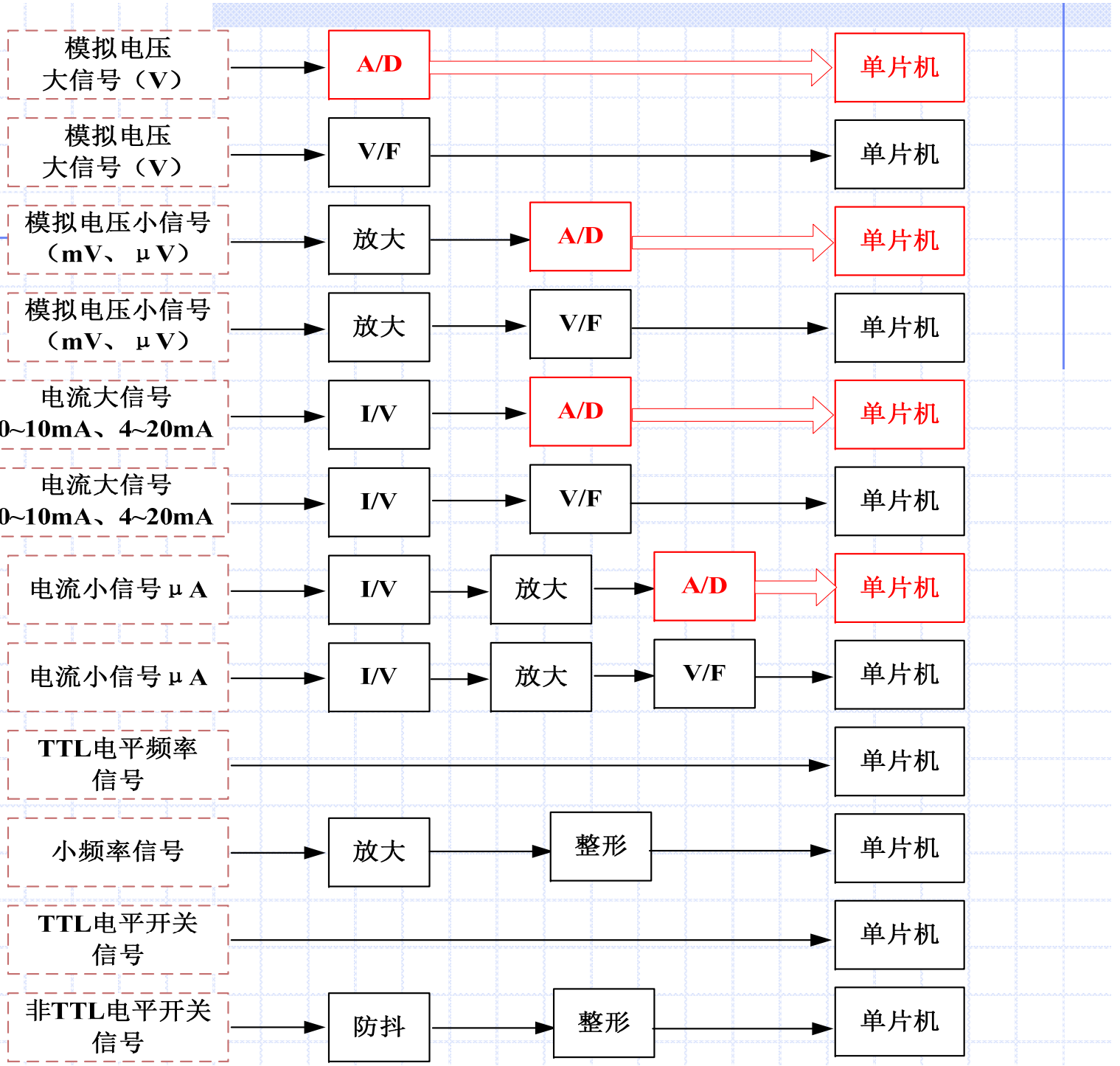
课程安排：

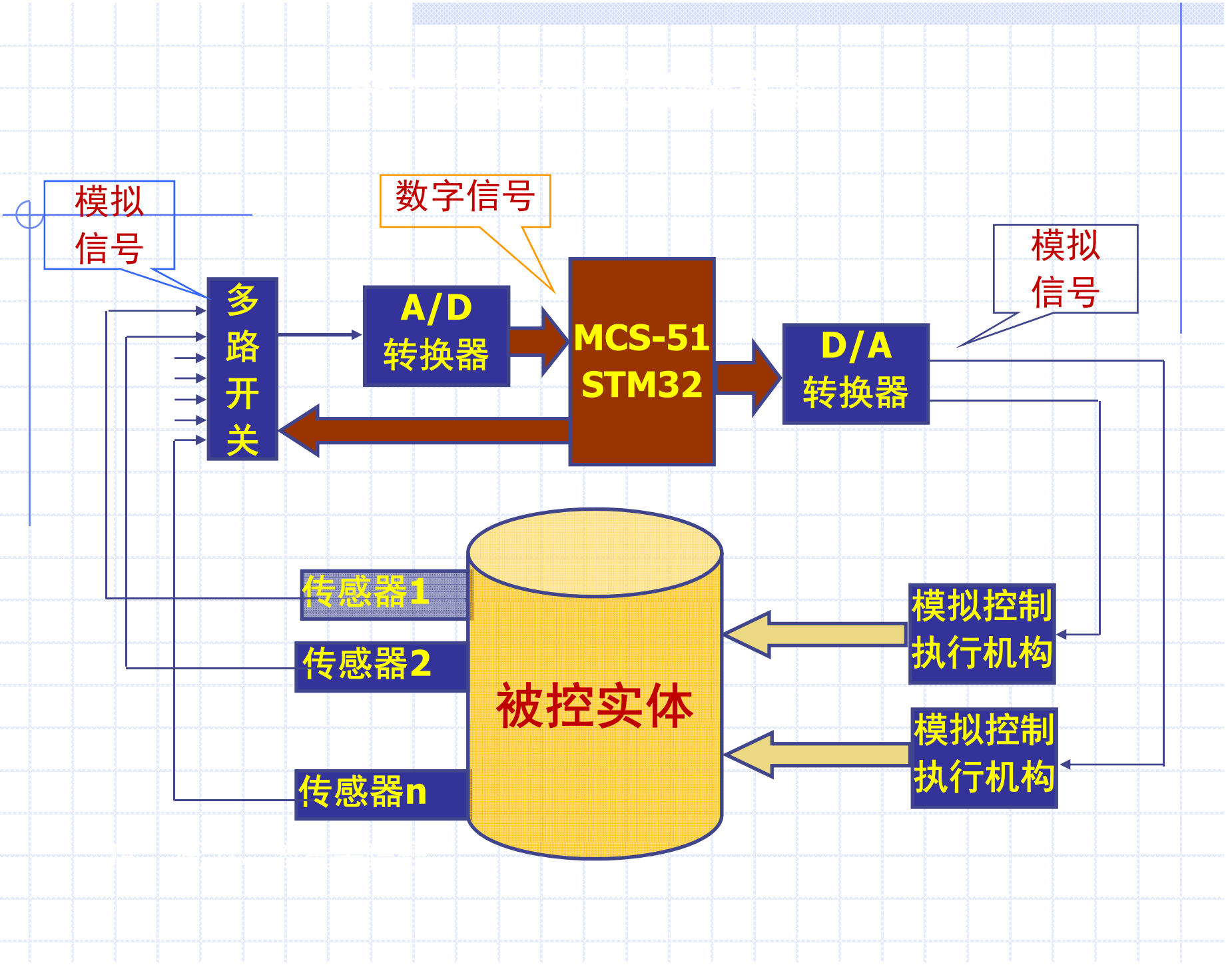
- §1 技术背景
- §2 实验板系统
- §3 MCS-51系统设计
- §4 STM32系统设计
- §5 采用GPIO的系统

嵌入式系统的组成和应用



前向通道信号处理方式


模拟电压大信号（V）
A/D
单片机
模拟电压大信号（V）
V/F
单片机
模拟电压小信号（mV、μV）
放大
A/D
单片机
模拟电压小信号（mV、μV）
放大
V/F
单片机
电流大信号0~10mA、4~20mA
I/V
A/D
单片机
电流大信号0~10mA、4~20mA
I/V
V/F
单片机
电流小信号μA
I/V
放大
A/D
单片机
电流小信号μA
I/V
放大
V/F
单片机
TTL电平频率信号
单片机
小频率信号
放大
整形
单片机
TTL电平开关信号
单片机
非TTL电平开关信号
防抖
整形
单片机


模拟
信号
数字信号
模拟
信号
多
路
开
关
A/D
转换器
MCS-51
STM32
D/A
转换器
传感器1
传感器2
传感器n
被控实体
模拟控制
执行机构
模拟控制
执行机构

一、89C51的实现方案

① 硬件组成

89C51单片机、

HY6264(8KB SRAM)、X2504(WDT/EEPROM)、

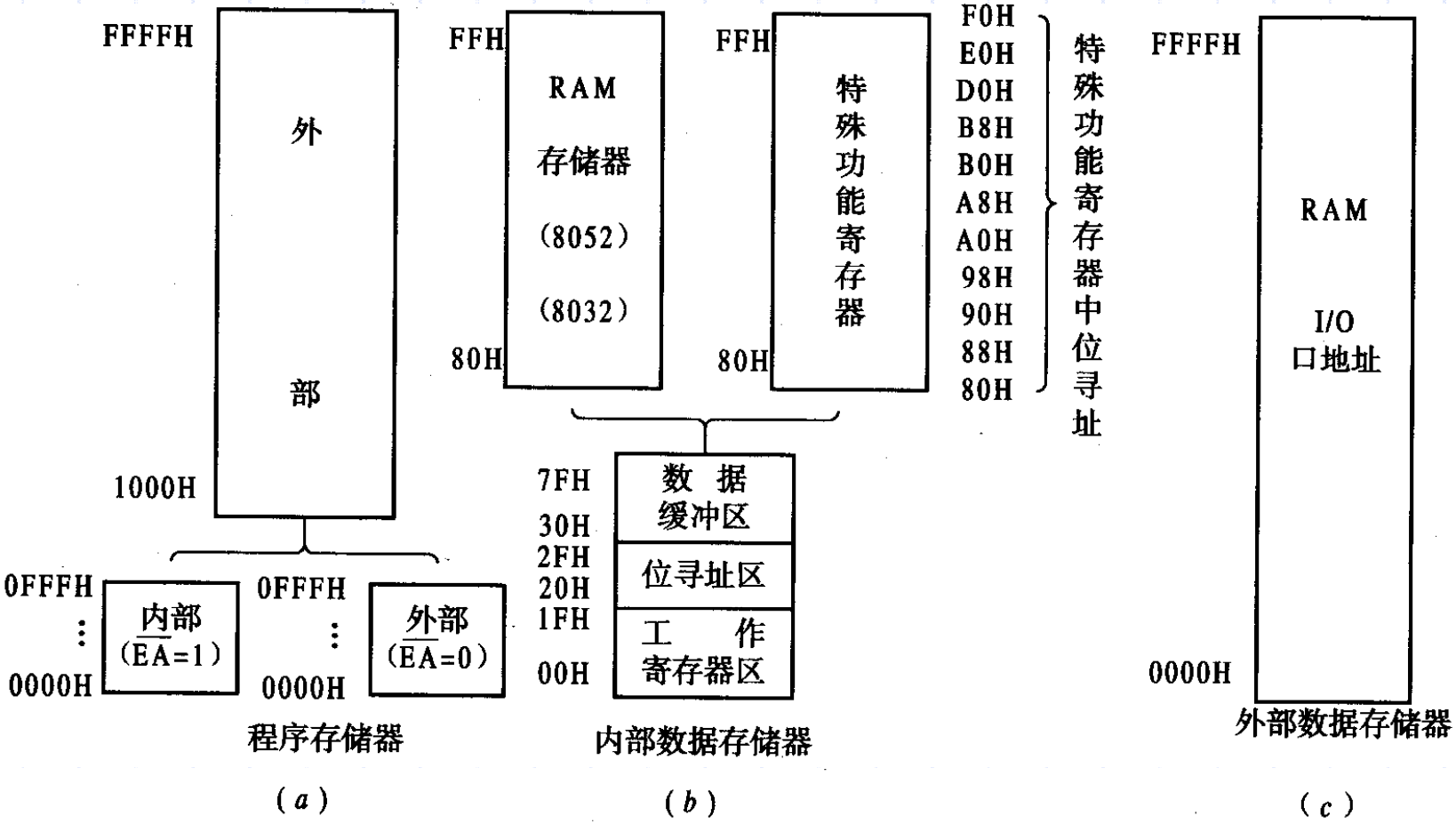
DS1302(时钟芯片)、75LBC184(485驱动器)、6N137(高速光耦)、6B595(输出驱动)、74HC138(译码)和74HC373(锁存器)等芯片。

② 方案评价：速度慢，芯片多

不同型号**MCS-51**单片机**CPU**处理能力和指令系统完全兼容，只是存储器和**I/O**接口的配置有所不同。

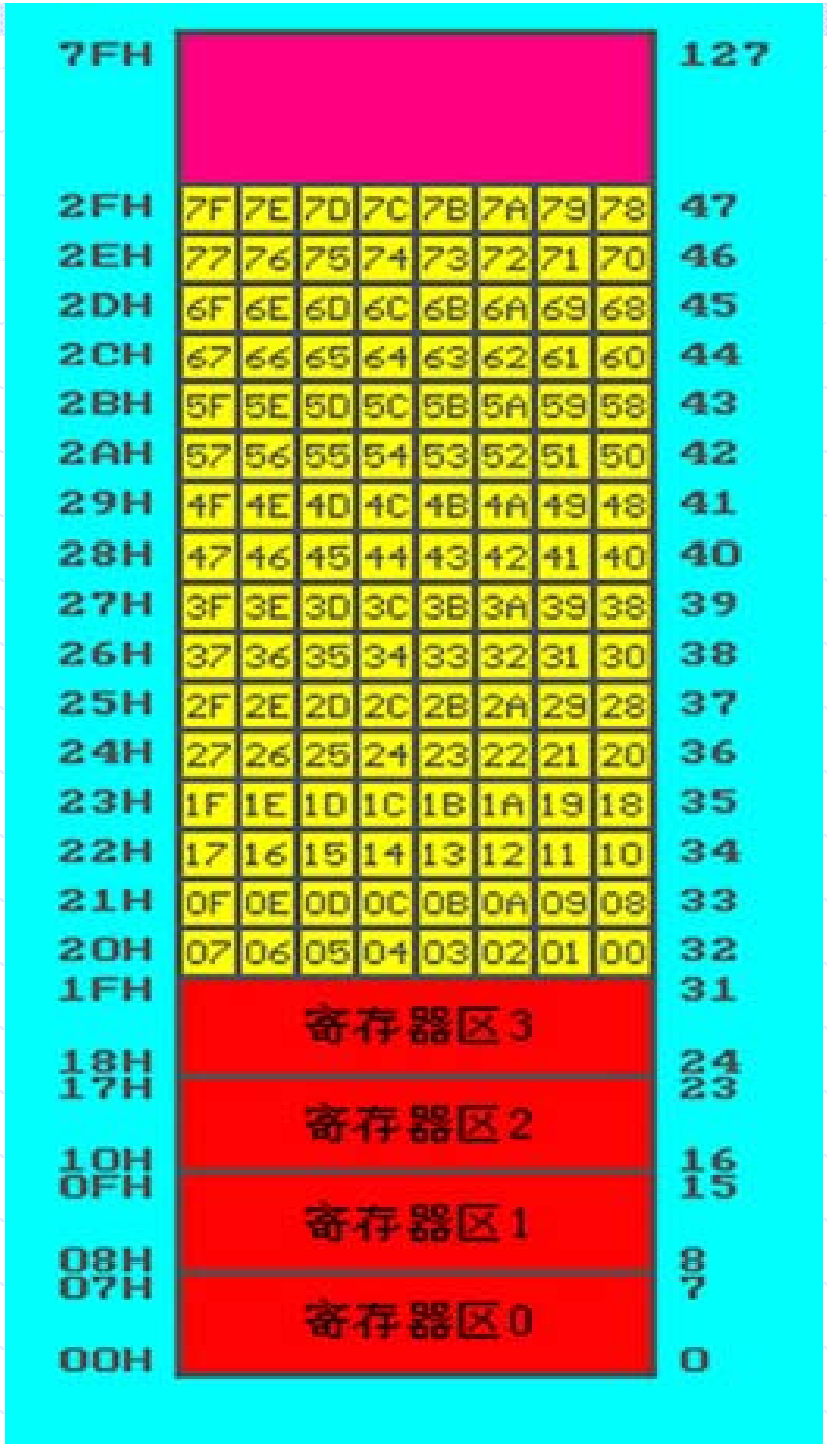
MCS-51单片机的组成：

1. **8**位**CPU**
2. 片内**ROM/EPROM**、**RAM**
3. 片内并行 **I/O**接口
4. 片内**16**位定时器/计数器
5. 片内中断处理系统
6. 片内全双工串行**I/O**口


FFFFH
外
部
1000H
0FFFH
内部
(EA=1)
0000H
0FFFH
外部
(EA=0)
0000H
程序存储器
(a)
FFH
RAM
存储器
(8052)
(8032)
80H
FFH
特殊功能寄存器
80H
F0H
E0H
D0H
B8H
B0H
A8H
A0H
98H
90H
88H
80H
特殊功能寄存器中位寻址
7FH
30H
数 据
缓冲区
2FH
20H
位寻址区
1FH
00H
工 作
寄存器区
内部数据存储器
(b)
FFFFH
RAM
I/O
口地址
0000H
外部数据存储器
(c)

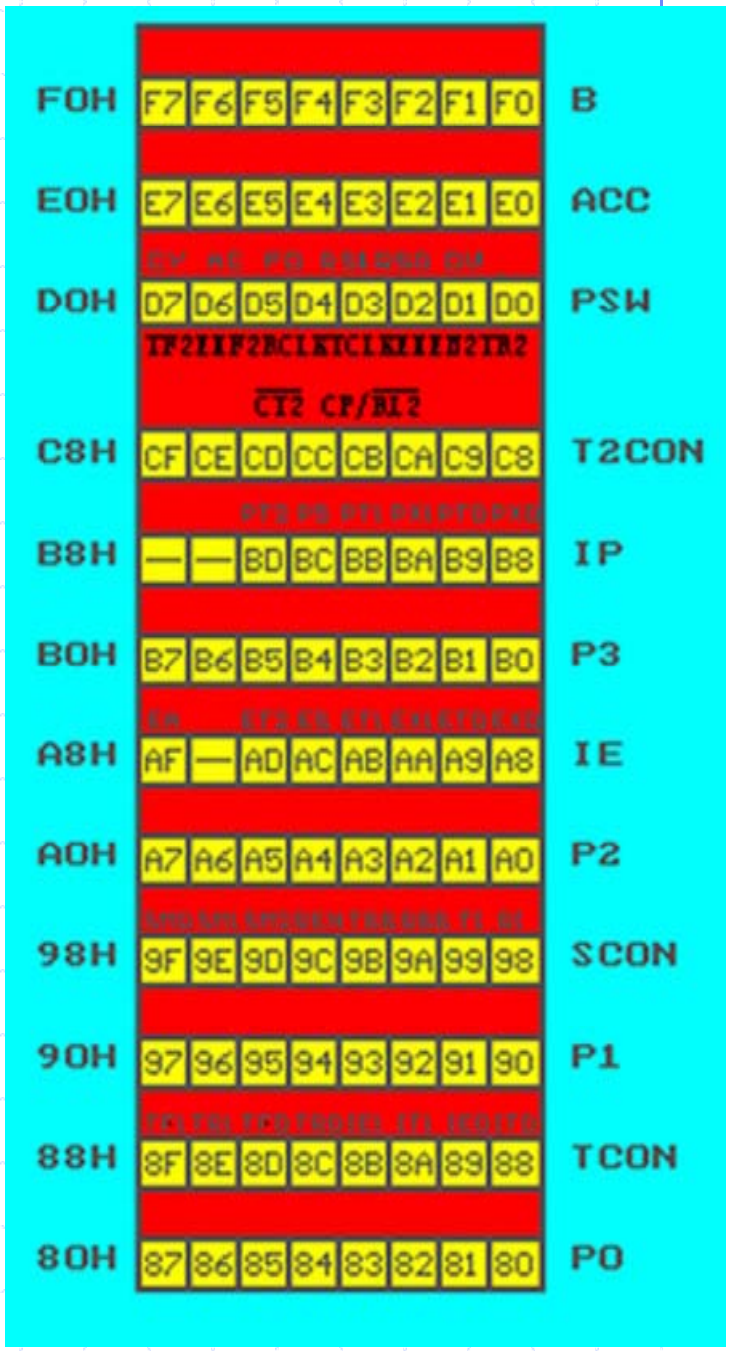
片内数据存储器

- 工作寄存器R0~R7
  - 00H～1FH
- 位寻址区
  - 20H～2FH
  - 位地址为：00H～7FH
- 用户RAM区
  - 30H～7FH



特殊寄存器

- 占用字节地址：80H～FFH
- 位寻址寄存器：
  - 其字节地址可被8整除。
- 专用寄存器：
  - A、B、PSW、DPTR、SP
  - TMOD、TCON、SCON …
- I/O接口寄存器：
  - P0、P1、P2、P3、SBUF



二、STM32微处理器的选择

选择STM32F103RC的理由：1个芯片可以替代多个芯片，芯片价格低，体积小(LQFP64)，集成度高(256K FLASH, 48K SRAM)，开发工具成本低(JTAG仿真器)。

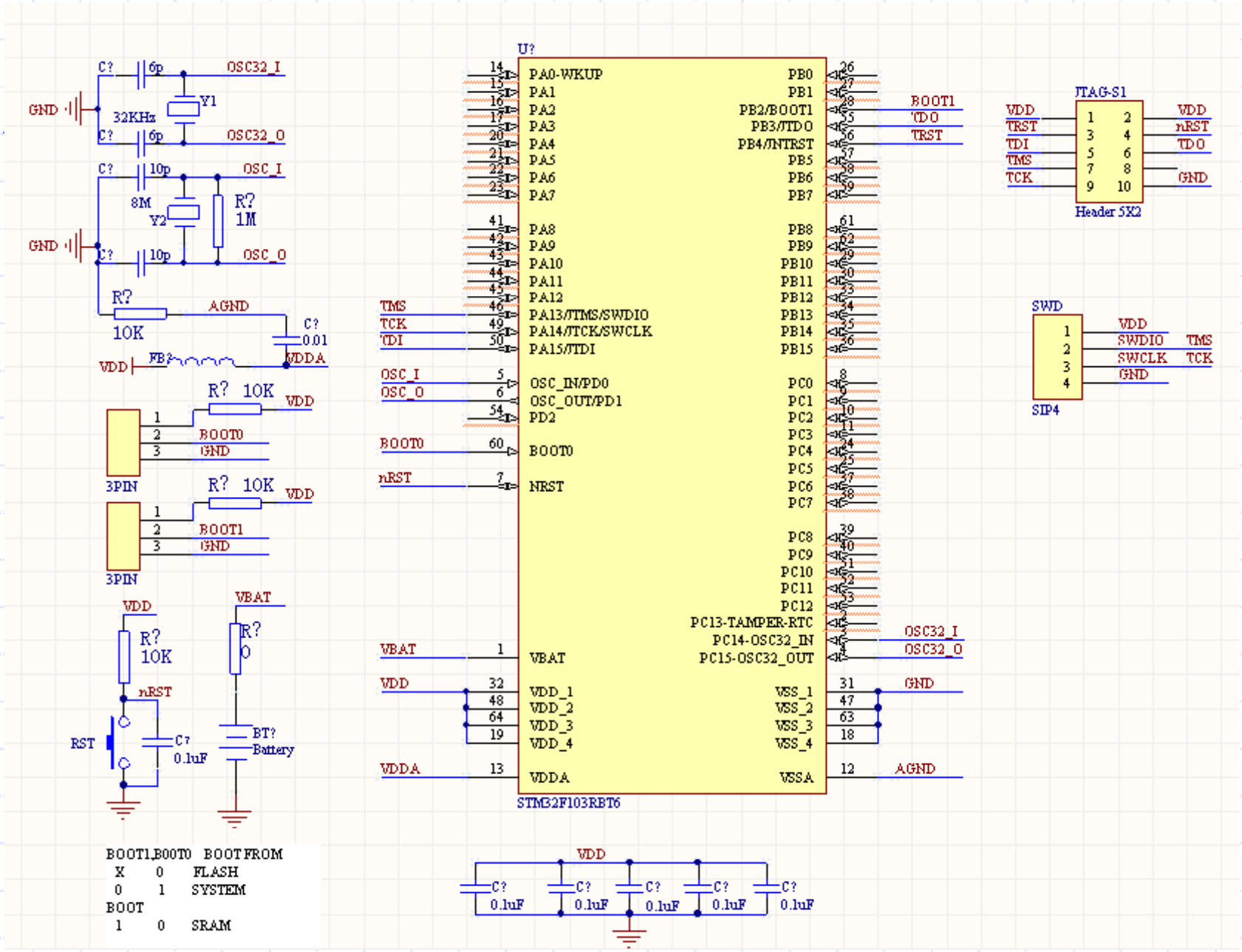
采用高集成度单片机STM32F103RC，更换MCS51，减少芯片数量，缩小体积，从而提高可靠性，降低功耗，增加SRAM的容量。

三、STM32F103RC的作用

STM32F103RC = 89C51 + SJA1000 + HY6264
+ 74HC138+74HC373+DS1302

CPU	32bit,72MHz	8bit ,40MHz/12
CAN	内含	外接 SJA1000 + 16MHz晶振
FLASH	256KB	8KB
SRAM	48KB	256B + 8KB(HY6264)
RTC	内含	外接DS1302

四、STM32F103最小系统



§1 技术背景

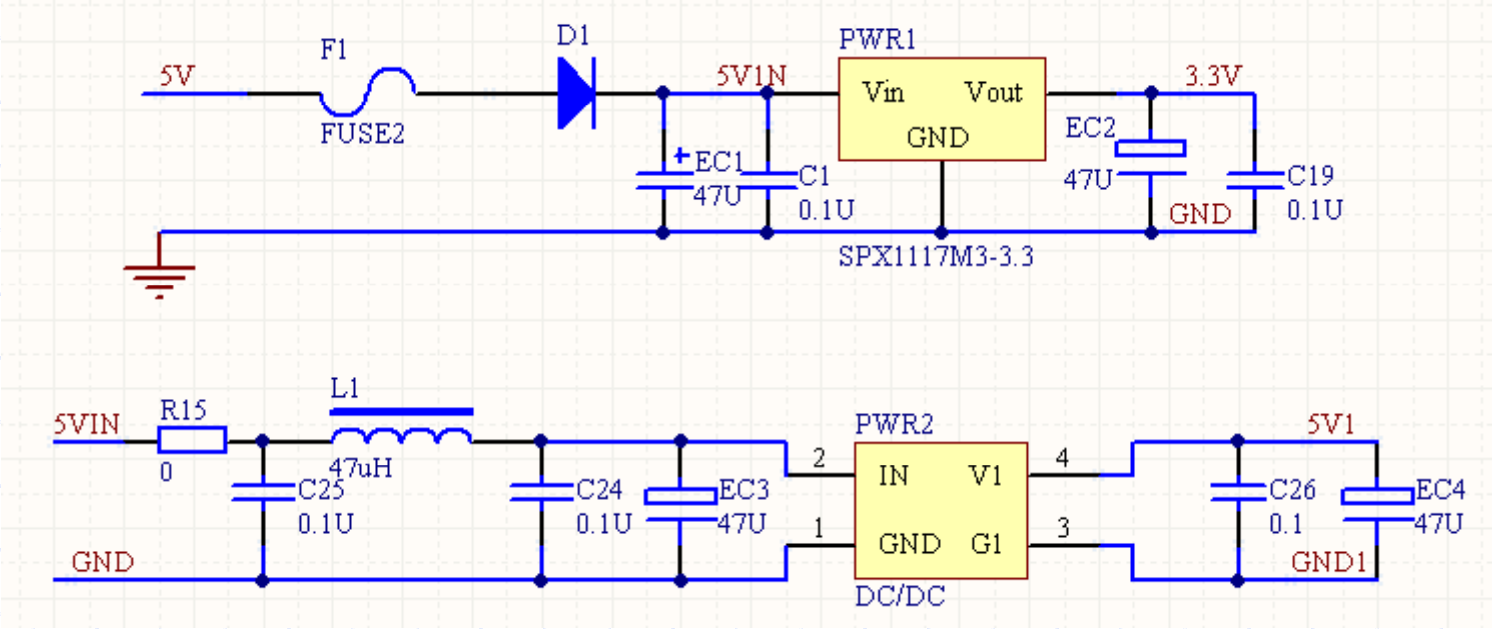
五、电源电路

① 芯片选择依据：电压、电流、稳压精度

② 电源芯片种类：

线性稳压(常规、低差压LDO)、开关式DC/DC

③ 隔离电源模块



课程安排：

- §1 技术背景
- §2 实验板系统
- §3 MCS-51系统设计
- §4 STM32系统设计
- §5 采用GPIO的系统

§2 实验板系统

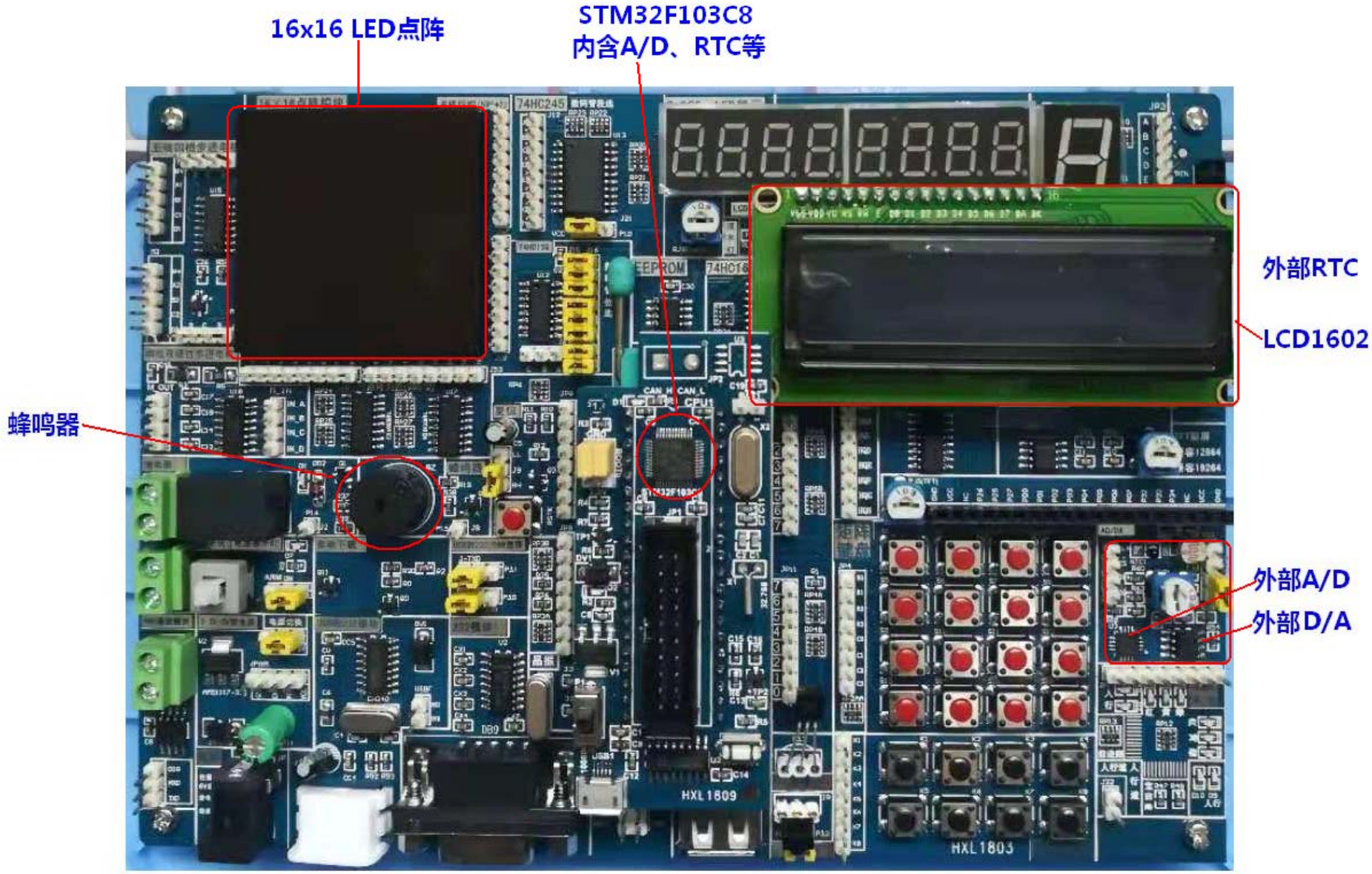
一、实验套件的开发目标

① MCU是半导体公司生产的，复杂程度较高，用户直接使用的难度较大。

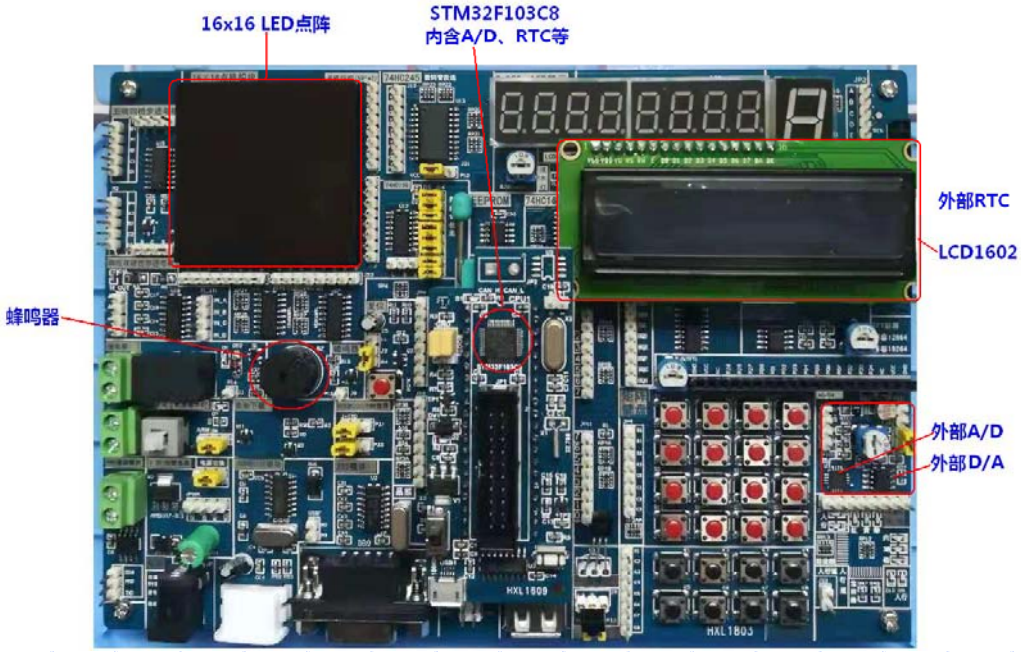
② 通常半导体公司会推出评估板（简单芯片）、开发套件（如MCU），充分展示MCU的功能，让用户测试和掌握，以降低用户使用该芯片的难度。

③ 普中MCS51+STM32套件开发目标：2种MCU的指令、MCU内部功能部件的使用方法、常用功能部件外部扩展和编程使用方法、以及专业的嵌入式软件开发环境（如KEIL）的应用。

二、实验板的组成


16x16 LED点阵
STM32F103C8
内含A/D、RTC等
外部RTC
LCD1602
蜂鸣器
外部A/D
外部D/A
HXL1609
HXL1803

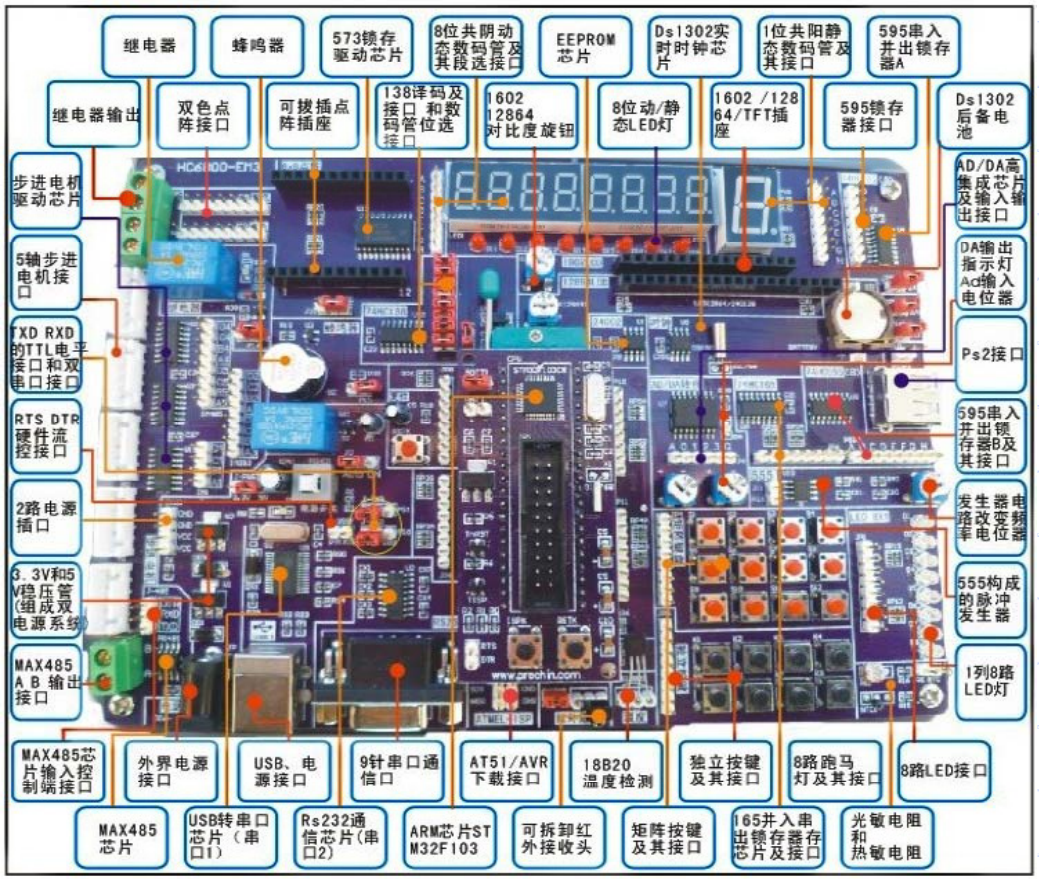
三、增强型STM32F103C8T6性能



1. STM32采用ARM32位Cortex-M3的CPU,工作频率可以达到72M，CPU速达1.25DMIPS/MHZ
2. 0等待访问存储器
3. 单周期硬件乘法和硬除法-加快计算能力
4. 68K的FLASH
5. 20K的SRAM
6. 时钟、复位和供电管理
7. 0至3.6伏供电和I/O管脚
8. 上电/断电复位(POR / PDR)、可编程电压监测器PVD
9. 内嵌4至16MHz高速晶体振荡器
10. 内嵌经出厂调校的8MHz RC振荡器
11. 内部40kHz的RC振荡器
12. PLL供应CPU时钟
13. 带校准的32kHz RTC振荡器
14. 低功耗
    A. 3种省电模式：睡眠、停机和待机模式
    B. VBAT为RTC和后备寄存器供电
15. 调试模式： 串行线调试(SWD)和JTAG调试接口
16. DMA控制器
    A. 7通道DMA控制器
    B. 支持的外设：定时器、ADC、SPI、I2C和USART
    C. 1个12位模数转换器，1us转换时间(16通道)
17. 多达80个快速I/O口
    26/37/51/80个多功能双向5V兼容的I/O口
    所有I/O口可以映像到16个外部中断
18. 多达7个定时器
    多达3个16位定时器，每个定时器有多达4个用于输入捕获 / 输出比较 / PWM或脉冲计数的通道
    16位6通道高级控制定时器

四、实验板的功能

功能简介



采用模块化分区，原理图清晰易懂，并带有注释，一看就懂 一学就会 一思就通

1 .双电源系统, 3.3V和5V可以自由切换。

2. 增强型32位ARM处理器，STM32F103C8T6。

3. STM32带有12位ADC，可以实现10路模拟输入。

4 . STM32带有2个硬I2C总线和2个硬SPI总线。

5. 5轴步进驱动，实现智能机器手模拟数控系统。

6 .工业用RS485通信模块，输出按工业标准。

7. 脉冲发生器模块，可以产生频率可调的方波。

8. 双串口独立模块，板载了2个串口。

9. TFT真彩屏和SD卡模块，用来显示文字、图片等。

10. IO输出扩展模块，可以扩展16个输出接口。

11. IO输入扩展模块，可以扩展8个输入接口。

12. 双色8*8点阵（红绿）及其驱动模块。

13. USB转串口模块，收发和硬件流控制端公开。

14. 16完全独立个LED灯，感受STM32的16位输出。

15. 支持ucOS II实时操作系统移植。

课程安排：

- §1 技术背景
- §2 实验板系统
- §3 MCS-51系统设计
- §4 STM32系统设计
- §5 采用GPIO的系统

课程安排：

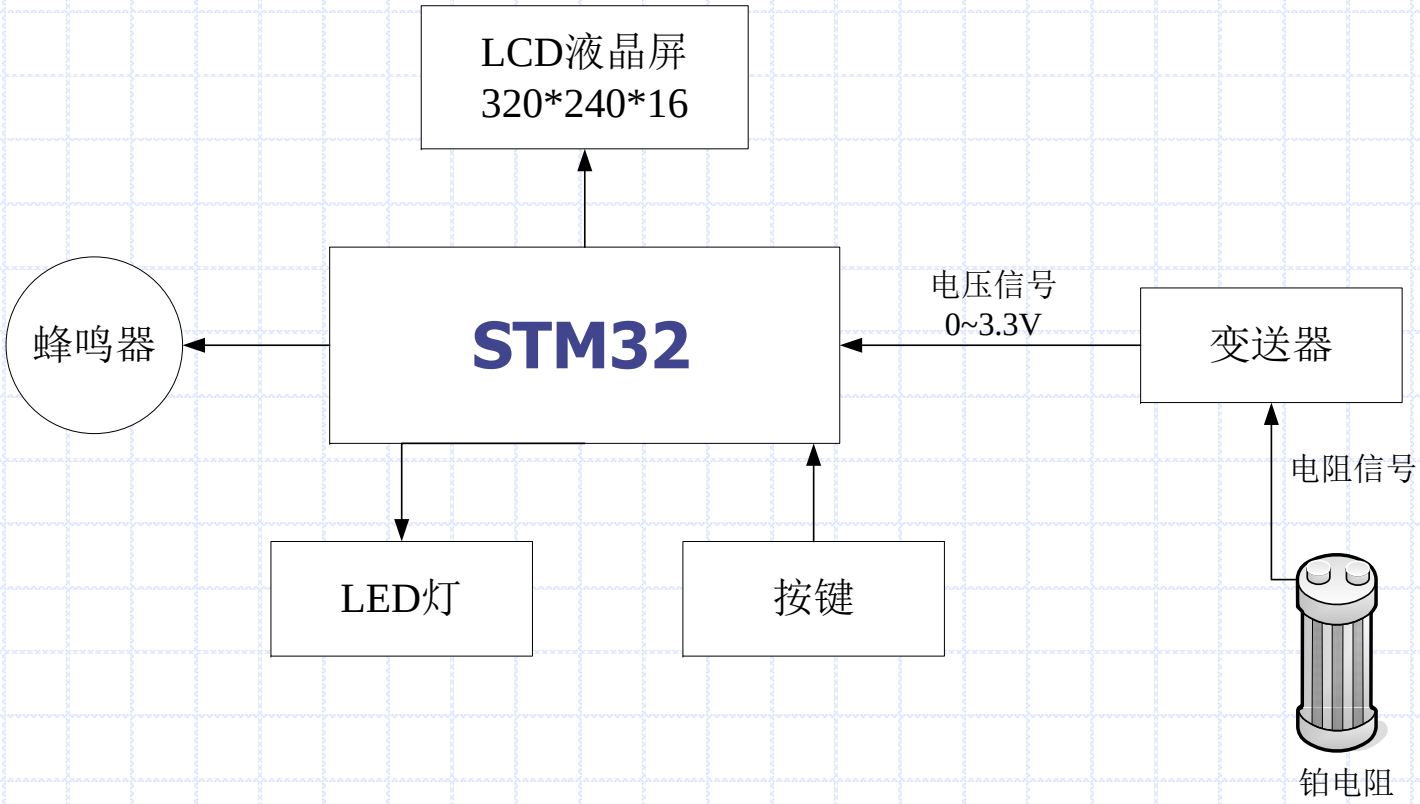
- §1 技术背景
- §2 实验板系统
- §3 MCS-51系统设计
- §4 STM32系统设计
- §5 采用GPIO的系统

STM32系统的设计

- 一、项目功能描述
- 二、系统组成
- 三、接口电路图设计
- 四、程序设计
- 五、运行结果
- 六、基于**STM32**的水质检测系统设计
- 七、基于**STM32**的室内温度控制系统
- 八、基于**RX23**的呼吸机控制系统

一、项目功能描述

- 温度监测仪可以实现如下功能：
  - 使用铂电阻传感器探测出目标温度，并通过图形的方式显示在LCD屏上
  - 当温度超过报警温度时，在LCD屏上显示报警状态
  - 当温度超过报警温度时，通过PWM控制蜂鸣器实现报警
  - 当温度超过报警温度时，点亮LED报警灯
  - 可以通过按键关闭或打开蜂鸣器及报警等功能
- 实例涉及到传感器技术、GPIO控制技术、中断技术、LCD控制技术、PWM控制技术、A/D转换技术等。

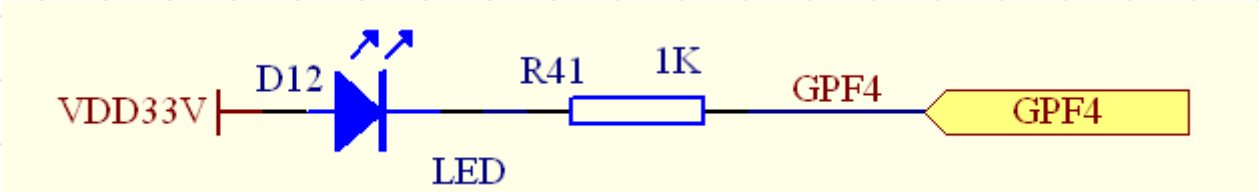


系统组成功能图

三、接口电路图设计

1、报警灯电路

- 处理器STM32通过GPF4控制报警LED灯，如图所示。

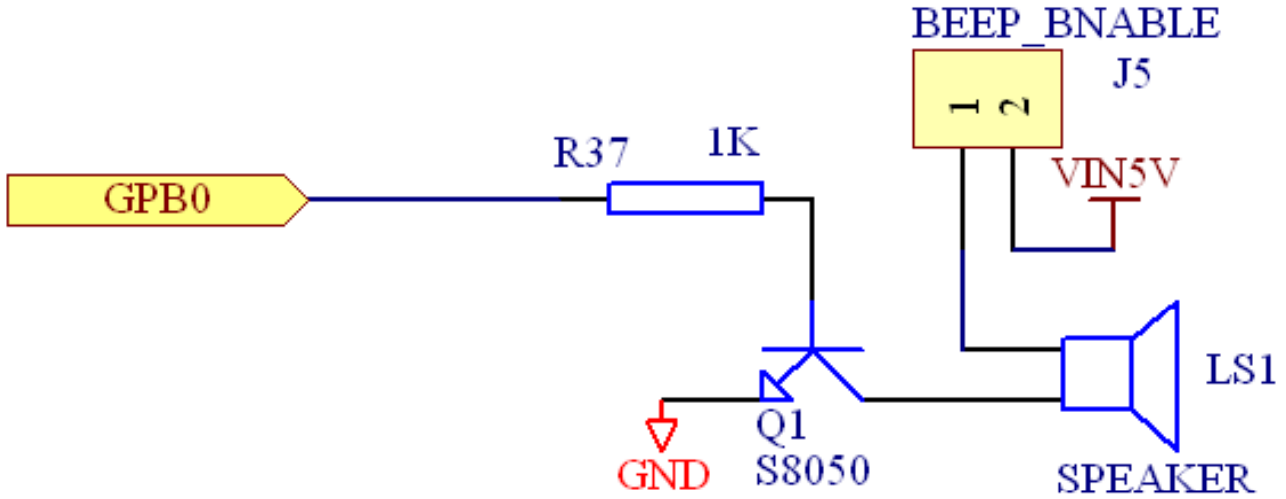


报警灯电路

三、接口电路图设计

2、蜂鸣器电路

- 处理器STM32通过GPB0输出PWM信号控制蜂鸣器，如图所示

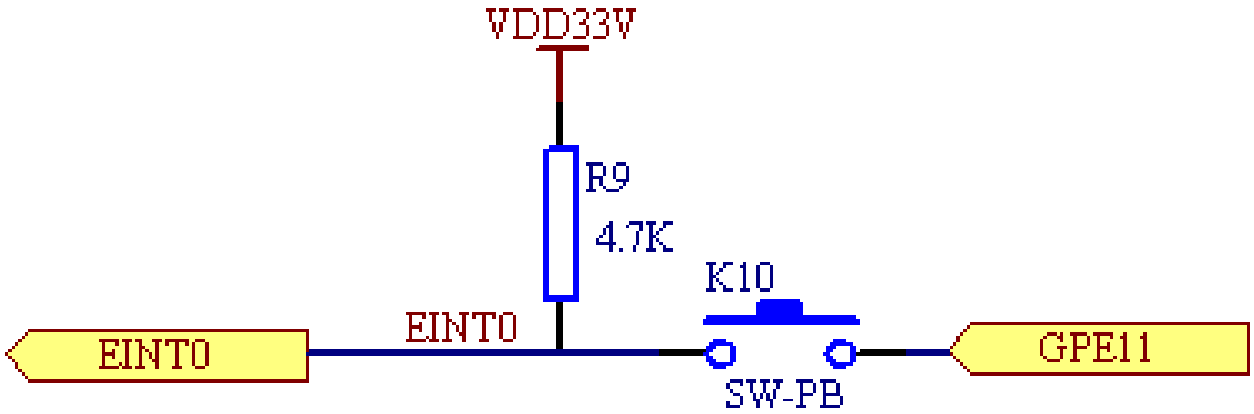


蜂鸣器电路

三、接口电路图设计

3、按键电路

- 当GPE11输出0电平时，按键按下后会在EINT0管脚上产生跳沿信号。系统利用此信号产生中断。如图所示



按键中断电路

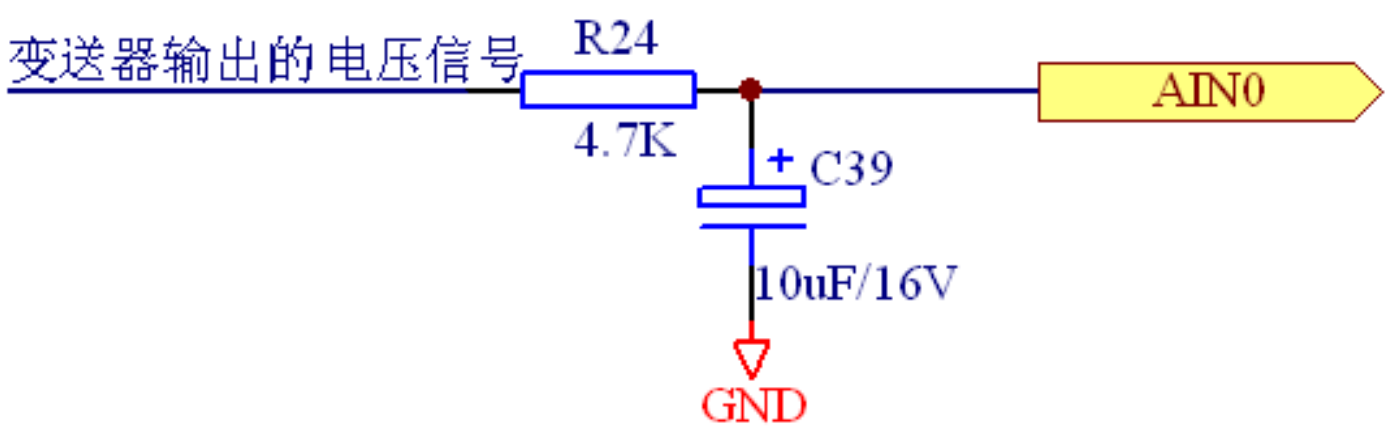
三、接口电路图设计

- 4、LCD液晶屏电路分为STN单色和TFT真彩色两种：
  - LCD屏的分辨率为320*240，颜色深度为16位真彩色TFT，或2行黑白LCD，电路接线参照实验。

三、接口电路图设计

5、温度采集电路

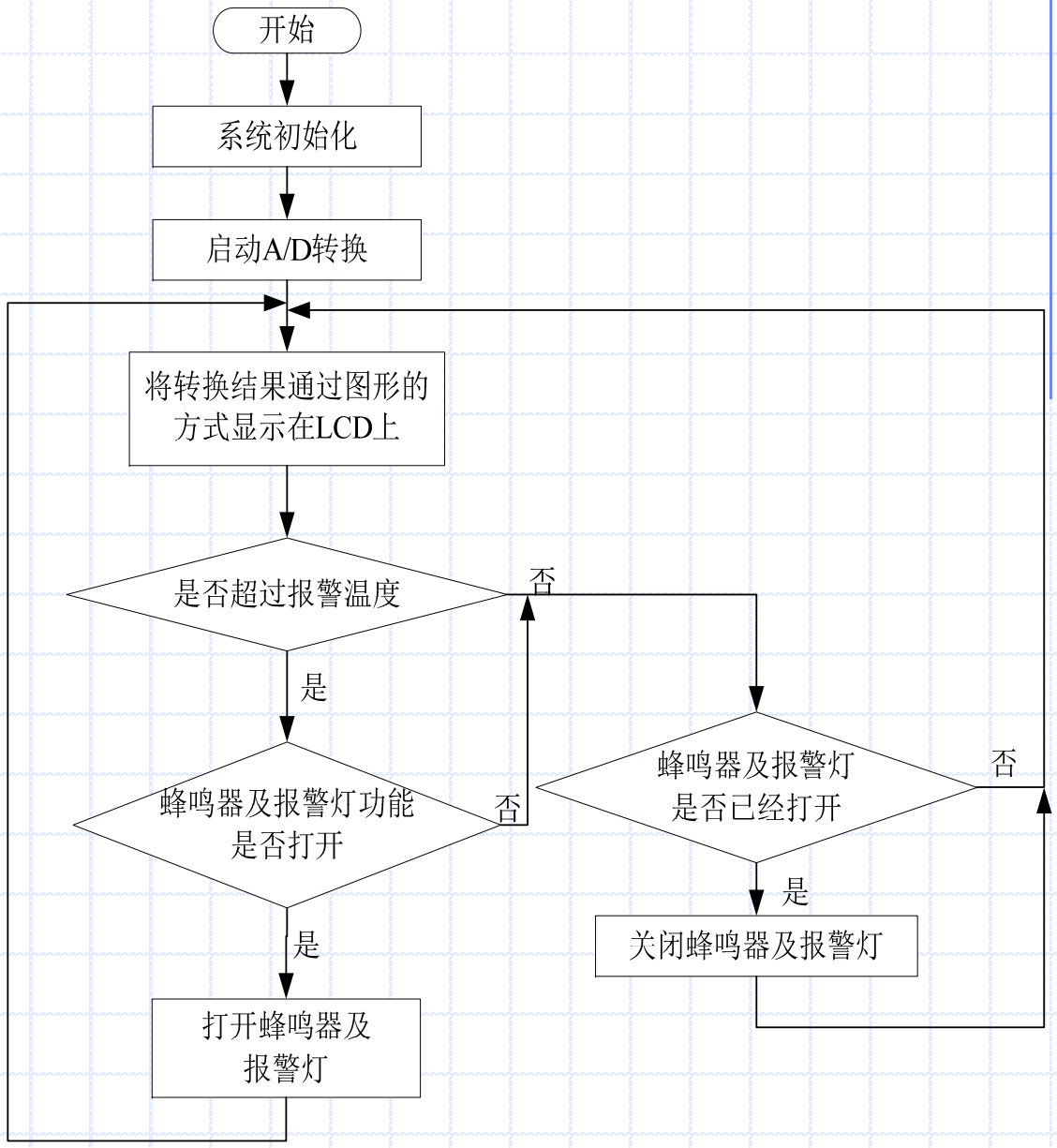
- 采样温度范围是0~100℃，变送器将温度信号转换为0~3.3V的电压信号。电压信号和S3C2410的AIN0相连。系统通过测量电压信



温度采集电路

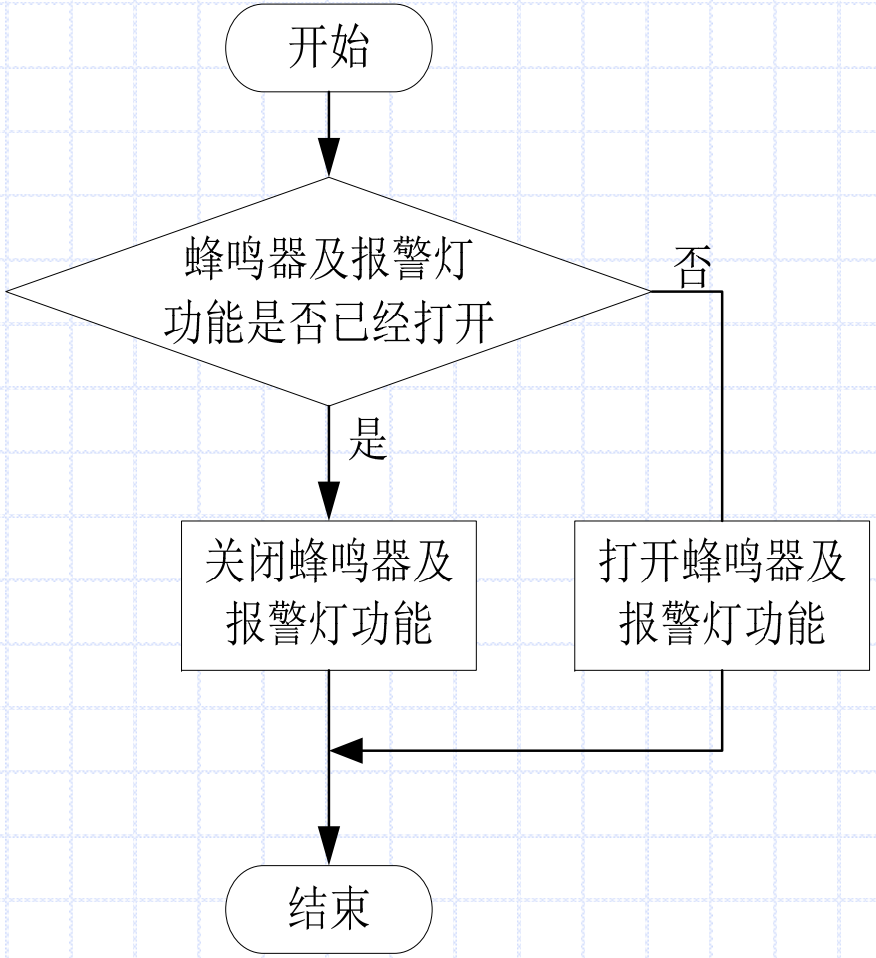
四、程序设计

1、程序工作流程



主程序流程

四、程序设计



按键中断处理程序流程

四、程序设计

2、主程序代码

```
1.  #include "2410lib.h"
2.   extern void  beep_gpio_setup(void);
3.   extern void  beep_start(void);
4.   extern void  beep_stop(void);
5.   extern UINT16T adc_get(void);
6.   extern void  lcd_init(void);
7.   extern void  int_init(void);
8.   extern void  led_init(void);
9.   extern void  led_on(void);
10.  extern void  led_off(void);
11.  extern void  Fill_area(int x0,int y0,int x1,int y1,unsigned short color);
12.  extern void  adc_init(void);
```

四、程序设计

```
13.     int flag_alarm=1;  //打开alarm功能的表示
14.     int flag_beep=0;   //蜂鸣器和报警灯打开标志
15.
16.     int main(int argc,char **argv)
17.{
18.      unsigned short adc_data;
19.      float f_value;
20.      sys_init();//初始化 s3c2410's Clock, MMU, Interrupt,Port and UART
21.      led_init();                    //LED接口初始化，配置对应的GPIO属性
22.      lcd_init();                    //初始化LCD控制器
23.      int_init();                    //初始化按键中断
24.      beep_gpio_setup();             //初始化PWM控制蜂鸣器
25.      adc_init();                    //初始化A/D控制器
```

四、程序设计

```
26.     while(1)
27.     {
28.         adc_data=adc_get();                    //获取到ad转换值
29.         f_value=adc_data*100.0/1024.0;         //转换为对应的电压值
30.         uart_printf(" %0.1f \n",f_value);
31.         Fill_area(60,80,260,120,0x0);          //填充温度计的默认颜色
32.         Fill_area(60,80,(int)(f_value*2),120,0xf000);
33.         delay(10000);
34.         if(f_value>80)                 //判断温度是否超过报警值80
35.          {
36.             if(flag_alarm)             //判断是否设置了报警功能
37.              {
38.                 beep_start();  //驱动蜂鸣器
```

四、程序设计

```
39.                         led_on();          //打开报警灯
40.                         flag_beep=1;       //置标志
41.                 }
42.                 else
43.                 {
44.                         if(flag_beep)      //判断标志
45.                         {
46.                                 beep_stop();     //关闭蜂鸣器
47.                                 led_off();       //关闭报警灯
48.                                 flag_beep=0;     //flag标志置0
49.                         }
50.                 }
51.         }
```

四、程序设计

```
52.                 else
53.                 {
54.                         if(flag_beep)                   //判断标志
55.                         {
56.                                 beep_stop();    //关闭蜂鸣器
57.                                 led_off();              //关闭报警灯
58.                                 flag_beep=0;    //flag标志置0
59.                         }
60.                 }
61.         }
62.         return 0;
63.   }
```

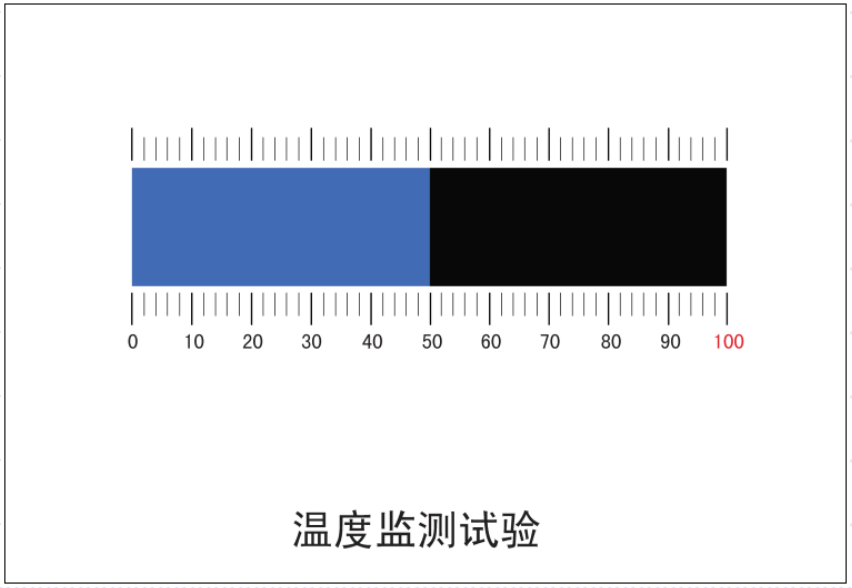
四、程序设计

3、按键中断处理程序代码

```
1. extern int       flag_alarm;
2. void  __irq int0_int(void)
3. {
4.     delay(3000);
5.     ClearPending(BIT_EINT0);
6.     uart_printf(" EINT0 interrupt occurred.\n");
7.         if(flag_alarm)
8.         {
9.                 flag_alarm=0;
10.        }
11.        else
12.                flag_alarm=1;
13.}
```

五、运行结果

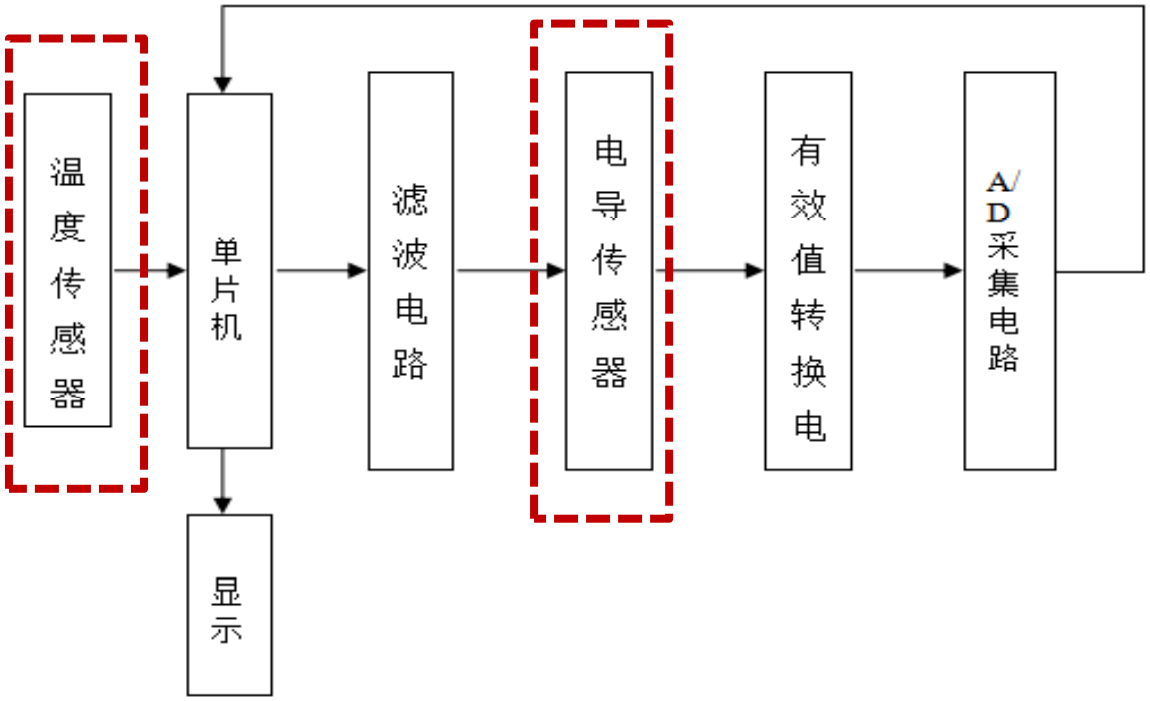
- 例程的运行后在液晶上的显示如图所示：



控制器温度监控界面

通过图形的方式直观的表示出温度情况。K10可以控制报警功能的开关，实现测试系统的各项功能。

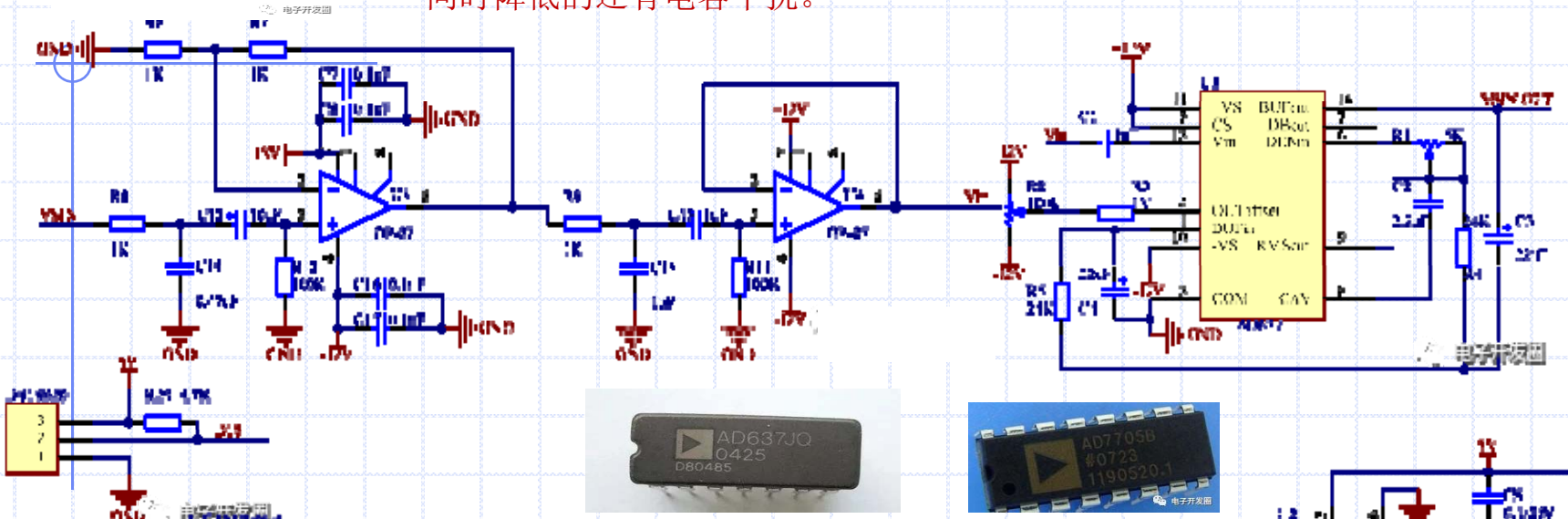
六、基于STM32的水质检测系统设计

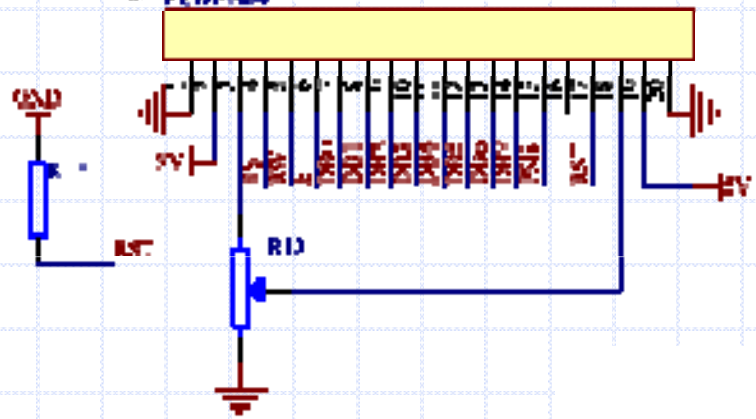


- 一个基于STM32F103RCT6单片机的地下水检测系统，该系统用于检测地下水的水质状况，可以体现水质状况的主要参数有水温，电导率，PH值还有浊度等。
- 对于水温来说，一般情况是地下水的水温越低，那么意味着，地下水所处的位置越深，即水位数据比较大，可以根据水温，大致的判断一下水位的情况，对于PH值来说，地下水的PH受地质环境影响非常大，绝大多数的地下水都是地表渗透下来的雨水等天然水，如果一个地区的土壤是碱性土壤，那么该地区的地下水绝大多数都会是弱碱性的水质。
- 对于电导率来说，这是衡量水质标准最主要的参数，除了纯水之外，其他的水都含有各种离子，例如Na+、Cu2+等，水的电导率能够反映水中离子的浓度，离子的浓度越大，则电导率就越高，正常的地下水电导率在一定范围之内。衡量地下水品质的另一标准就是浑浊程度，浑浊的程度是水中杂质含量的一个直观体现，甚至用肉眼就可以看出来，浑浊程度过高的地下水肯定是不能直接饮用的，这样的水质可以通过沉淀过滤消毒后才能饮用。

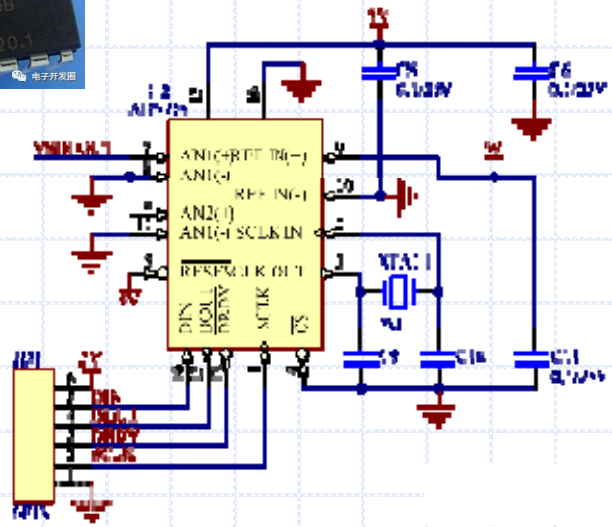
电导率传感器

本系统采用的是电导率电极DJS-1C铂黑电极，电导率电极使用的材料一般都是铂，为了减少极化效应，在镀铂黑时在铂表面上镀了一层蓬松的黑色金属铂。多孔的铂黑可以加大电极的表面积，这样电流的密度就会变小，极化的效应也会变低，同时降低的还有电容干扰。





◆AD转换芯片有很多，如AD0808、AD0809、AD574、AD7705。其中AD7705是AD公司推出的16位Σ-△(电荷平衡式)A / D转换器。



◆LCD12864是一种具有4 位/8 位并行、2 线或3 线串行多种接口方式；12864 汉字图形点阵液晶显示模块，可显示汉字及图形，内置 8192 个中文汉字（16X16 点阵）。

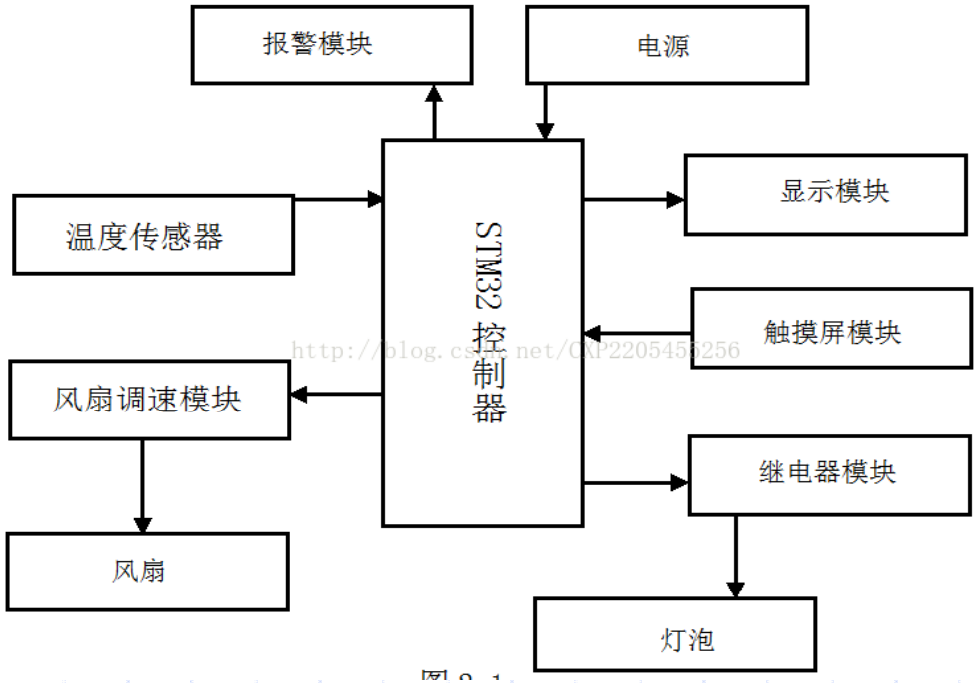
◆**系统软件的开发**

◆系统采用C语言编程语言，在Keil开发环境中运行，

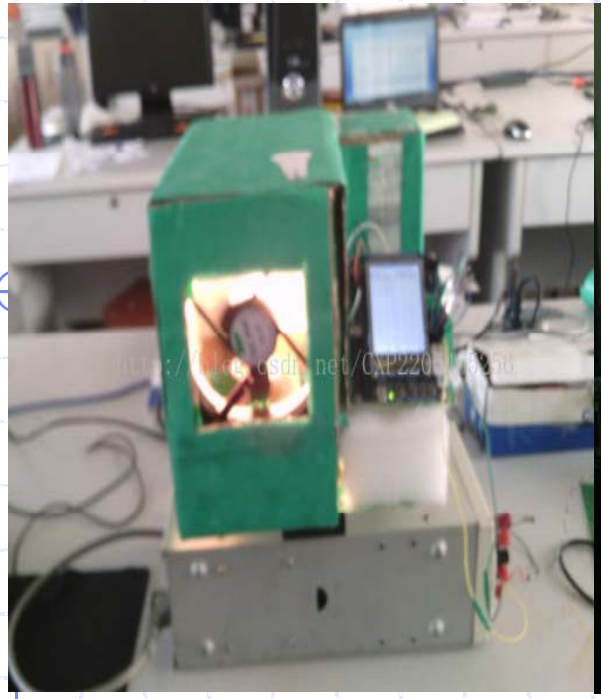
七、基于STM32的室内温度控制系统

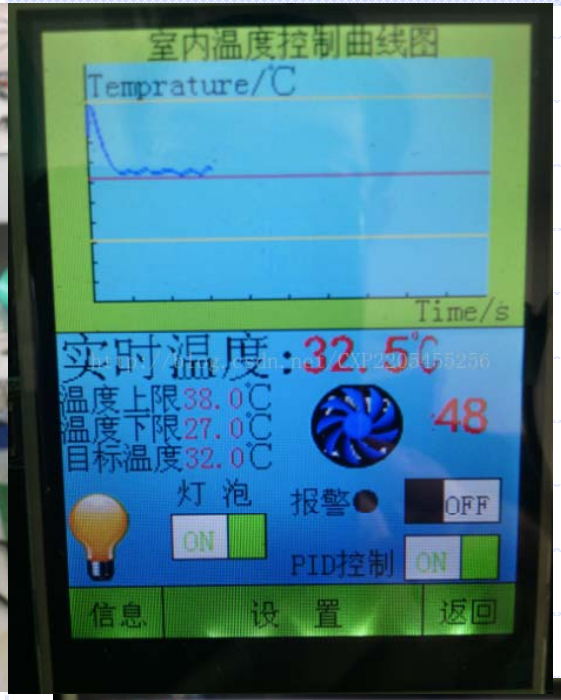
设计基于DS18B20的数字式室内变电站温度控制系统，控制程序主要包括主程序、读出温度子程序、按键子程序、控制风机子程序等。要求能检测、显示及通过控制排气风机来降低室内温度。

七、基于STM32的室内温度控制系统



- 采用的温度传感器是DS18B20数字温度传感器，当温度变化时，该传感器通过内部的模拟信号通过A/D转换器，将温度数据转换为数字信号，单线协议发送至单片机端口，单片机处理温度数据后通过TFT彩色液晶显示，同时单片机将温度信号分别与控制温度、报警温度比较，并通过PID控制算法计算出控制量调节风扇转速，从而达到控制室内温度的目的。提供丰富多彩的GUI人机交流界面，和灵敏的触摸屏输入模式，具有显示真实直观、简洁高效的显示菜单。




室内温度控制曲线图
Temprature/℃
Time/s
实时温度：32.5℃
温度上限38.0℃
温度下限27.0℃
目标温度32.0℃
灯 泡
ON
报警
OFF
PID控制
ON
48
信息
设 置
返回


设定值：28
Kp:80.5 Ki:00.0 Kd:00.0
时间 PWM 温度 时间 PWM 温度
218 65 28.8 252 60 28.7
220 60 28.7 254 60 28.7
222 55 28.6 256 55 28.6
224 60 28.7 258 60 28.7
226 60 28.7 260 65 28.8
228 60 28.7 262 65 28.8
230 60 28.7 264 60 28.7
232 65 28.8 266 55 28.6
234 65 28.8 268 60 28.7
236 65 28.8 270 65 28.8
238 60 28.7 272 65 28.8
240 60 28.7 274 70 28.8
242 55 28.6 276 75 28.9
244 60 28.7 278 70 28.8
246 55 28.6 280 60 28.7
248 55 28.6 282 60 28.7
250 55 28.6
Cortex-M3 STM32


室内温度控制参数设置
目标温度
32.0℃
上限温度
38.0℃
下限温度
27.0℃
参数确定
参数确定后，系统将按照新的参数指标控制。
参数取消
参数取消后，参数指标不更新，系统控制进行
Cortex-M3 STM32

Kp可以说是PID算法中必不可少的一个量。没有它控制就不可能实现。它的作用是根据当前量与设定量的差值按照一定比例放大后得到控制量。

```
//pid.h
#ifndef __PID__
#define __PID__
/*PID = Uk +KP*[E(k)-E(k-1)]+KI*E(k)+KD*[E(k)-2E(k-1)+E(k-2)];      (增量型PID算式)
函数入口: RK(设定值),CK(实际值), KP, KI, KD
函数出口: U(K)*/
typedef struct PIDValue
{
   int8 KP;
   int8 KI;
   int8 KD;
   int8 F;
   int8 BITMOV;
   int EK[3];
   int UK;
   int RK;
   int CK;
   int UK_REAL;
}pid_str;
//PIDValueStr  PID;
void    pid_exe(pid_str*PID)  ;
#endif
```

```
//pid.c
/*PID = PID->UK_REAL +PID->KP*[E(k)-E(k-1)]+PID->KI*E(k)+PID->KD*[E(k)-2E(k-1)+E(k-2)];(增量型PID)
函数入口:PID->RK(设定值),PID->CK(实际值),PID->KP,PID->KI,PID->KD
函数出口: U(K)*/
#include"defines.h"
#include"pid.h"
#define MAXOUT 0xff
//#define MAXGAP 100
void pid_exe(pid_str*PID)
{
  PID->EK[2]=PID->EK[1];
  PID->EK[1]=PID->EK[0];
  PID->EK[0]=PID->RK-PID->CK;
  PID->UK_REAL=PID->UK_REAL
    +PID->KP*(PID->EK[0]-PID->EK[1])//微分一次后积分即原数
    +(float)PID->KI*PID->EK[0]/PID->F//直接积分
    +(float)PID->KD*(PID->EK[0]-2*PID->EK[1]+PID->EK[2])*PID->F;//二阶微分后积分即一阶微分
  if((PID->UK_REAL>>PID->BITMOV)>=MAXOUT)
  {
    PID->UK=MAXOUT;
  }elseif(PID->UK_REAL>>PID->BITMOV<=-MAXOUT)
  {
    PID->UK=-MAXOUT;
  }else
  {
    PID->UK=PID->UK_REAL>>PID->BITMOV;
  }
}
```

代码用到的是增量型的PID（即UK_REAL +PID->KP*[E(k)-E(k-1)]+PID->KI*E(k)+PID->KD*[E(k)-2E(k-1)+E(k-2)];这句话所对应的是pid控制量在之前pid控制量的基础上增加的值，相当于求了一次导）。

最终输出的结果将每一次运算的值累加输出就行了。

八、基于RX23的呼吸机控制

- ルネサス：新型コロナウイルス感染症と闘うため、公開済みオープンソースの設計仕様に基づいた人工呼吸器に向け、リファレンスデザインを作成
- 2020/04/16
- Motor Fan illustrated編集部

◆ルネサス エレクトロニクスは、医療用人工呼吸器アプリケーションを効率的に短期間で設計できるよう、オープンソースの設計仕様に基づいた人工呼吸器向けにリファレンスデザイン（参照設計）を作成した。

◆　新型コロナウイルス（COVID-19）感染症が拡大し続け、多くの地域で患者が病院の収容能力を上回り、人工呼吸器が大幅に不足するという事態が発生している。「そこでルネサスのエンジニアは、新型コロナウイルスのパンデミック（世界的流行）と闘う中で世界が直面する課題に対処するために、人工呼吸器システムのリファレンスデザインを作成しました。ルネサスの幅広い製品ポートフォリオとシステム設計のノウハウを活かすことにより、病院や自宅での操作が可能な医療用人工呼吸器システムを短期間で開発できるよう支援します」と、ルネサスのIoT・インフラ事業本部、執行役員のChris Allexandre氏は述べている。

　ルネサスは、メドトロニック社のPB560などのいくつかの設計仕様が公開されているオープンソースの人工呼吸器システムを基に、人工呼吸器用のリファレンスデザインを用意した。これは、患者の状態を監視しながら、患者に送られる1回のガスの量や配分量をコントロールするもの。この人工呼吸器は持ち運び可能なため、ガスタンクの有無にかかわらず使用することができる。さらに、加湿器を人工呼吸器の摂取経路に接続して患者の呼吸を和らげ、長時間接続することで、より効率的に快方に向かわせることができる。

　リファレンスデザインは、約20個のルネサスの半導体を使用している。人工呼吸器内のデジタル信号のやり取りをつかさどるマイコンや電源IC、アナログICなどで構成されている。この人工呼吸器のシステムは、センサ基板とモータ制御基板を実装しており、Bluetooth接続が可能。これにより、医療従事者はタブレットや携帯機器を使って、複数の患者を同時に監視することができる。各基板には、マイコンが搭載されており、接続された各基板の状態を監視し、指令を制御する。また、この人工呼吸器用ソリューションは、各機能をチェックするシステムを採用しているため、規制に関する認可を容易にし、患者への安全を提供する。

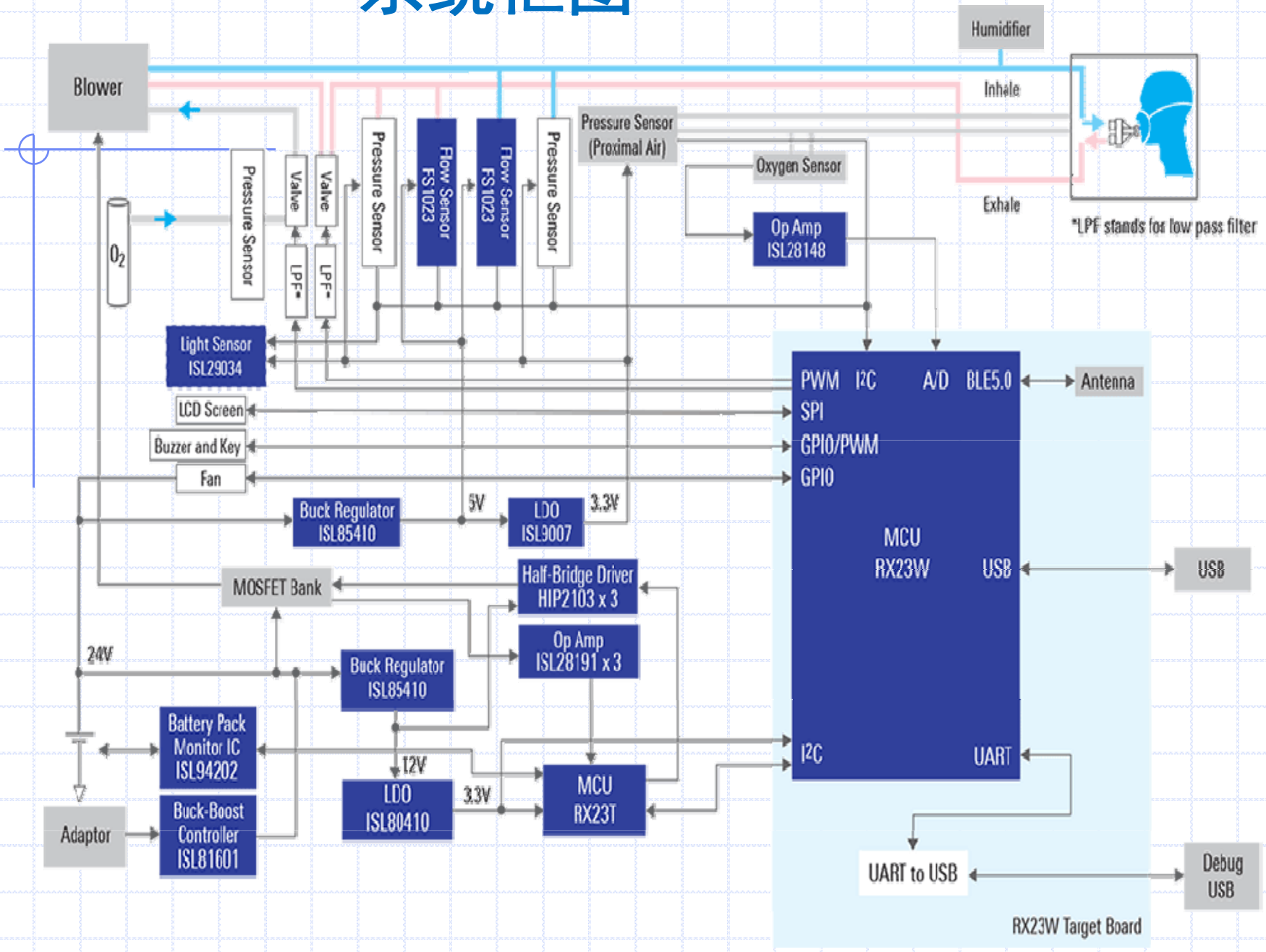
◆詳細については、ルネサスの人工呼吸器向けリファレンス

呼吸机系统

- 新冠肺炎大流行在全球造成了严重破坏，越来越多的人住进了医院。患者对呼吸机的需求量非常大，因此医生正在努力制作简易呼吸机来满足需求。许多公司通过各种方式为这项事业做出了巨大贡献，包括开放他们的知识产权。瑞萨电子也通过提供呼吸机组装电路板的参考设计，为这项全球事业提供帮助。
- 这是一个呼吸机系统参考设计，可以提供在走廊或非 ICU 病房使用的便携式呼吸机。本机可在辅助控制和压力控制两种模式下为患者提供高压氧气。辅助控制模式为病人每次吸气提供一定的气量。流量传感器 (FS1023) 可以监测吸气管处的气体流量，而潮气量则由 MCU 使用流速对时间的积分进行计算。氧气阀由 MCU 控制，控制氧气比。压力控制模式为患者每次吸气提供一定的压力。面罩上连接了一个近端空气压力传感器，可以监测吸气压力，并将这一信息发送给 RX23W MCU。用于提供压力及将空气鼓入系统的鼓风机由 RX23T MCU 控制的电机控制板驱动。RX23T和RX23W通过I2C进行通信。同时，在系统中增加了一个加湿器，用于为患者提供温润的空气。
- 该系统使用了两个 MCU，以提高安全性，因为这些设备可以相互监测和重置。

- 系统优势
  - 安全系统使用两个 MCU，可以相互监测
  - 系统支持辅助控制模式和压力控制模式
  - 硬件能够提供低潮气量、峰值压力、断开连接和呼吸暂停警报
  - FS1023 可以监测气量和流速
  - MCU 控制的阀门可以控制氧气比，流量传感器可监测氧气比
  - 呼气压力由 MCU 通过呼气阀控制
  - 氧气传感器用于检测 FiO2（吸入氧浓度）
  - 鼓风机由压力/流量传感器反馈控制
  - LCD 亮度由监测环境光的光传感器控制
  - 采用电池平衡监测器以延长电池寿命

系统框图


Humidifier
Blower
Inhale
Exhale
*LPF stands for low pass filter
Pressure Sensor (Proximal Air)
Oxygen Sensor
Op Amp ISL28148
Pressure Sensor
Flow Sensor FS1023
Flow Sensor FS1023
Pressure Sensor
Valve
Valve
LPF
LPF
O_2
Light Sensor ISL29034
LCD Screen
Buzzer and Key
Fan
PWM I2C A/D BLE5.0
Antenna
SPI
GPIO/PWM
GPIO
MCU RX23W
USB
USB
Buck Regulator ISL85410
5V
LDO ISL9007
3.3V
MOSFET Bank
Half-Bridge Driver HIP2103 x 3
Op Amp ISL28191 x 3
24V
Buck Regulator ISL85410
Battery Pack Monitor IC ISL94202
12V
LDO ISL80410
3.3V
MCU RX23T
I2C
UART
Adaptor
Buck-Boost Controller ISL81601
UART to USB
Debug USB
RX23W Target Board

推荐产品

产品	描述	相关文档	订购
微控制器			
RX23W	32 位 MCU，带 Bluetooth® 5 ，用于物联网终端设备、系统控制和在单个芯片上实现的无线通信	数据手册	选择特定设备
RX23T	内置浮点运算单元 (FPU) 的微控制器，最适合控制单个逆变器	数据手册	选择特定设备
LDO			
ISL9007	低 IQ 和高 PSRR 的高电流 LDO	数据手册	购买/样品
ISL80410	40V、低静态电流、150mA 线性稳压器	数据手册	购买/样品
降压稳压器			
ISL85410	宽 VIN 1A ， 1A 同步降压稳压器	数据手册	购买/样品
电池组监控器 IC			
ISL94202	独立式 3 至 8 芯锂离子电池组监控器	数据手册	购买/样品
升降压控制器			
ISL81601	60V 双向四开关同步升降压控制器	数据手册	购买/样品
传感器			
FS1023	气体和液体流量传感器模块	数据手册	购买/样品
ISL29034	集成式数字光传感器	数据手册	购买/样品
半桥驱动器			
HIP2103	60V、1A/2A 峰值、半桥驱动器（带 4V UVLO）	数据手册	购买/样品
运算放大器			
ISL28148	4.5 MHz，带超低输入偏置电流的单路精密轨对轨输入-输出 (RRIO) 运算放大器	数据手册	购买/样品
ISL28191	单路电源超低噪声，超低失真，轨到轨输出运算放大器	数据手册	购买/样品

课程安排：

- §1 技术背景
- §2 实验板系统
- §3 MCS-51系统设计
- §4 STM32系统设计
- §5 采用GPIO的系统

主要内容

- 一、 GPIO结构
- 二、 GPIO寄存器
- 三、 GPIO函数
- 四、 GPIO编程举例
- 五、 练习题

GPIO的功能

（1）最基本的功能是可以驱动**LED**、产生**PWM**、驱动蜂鸣器等。

（2）具有单独的位设置或位清除，编程简单。端口配置好后，只需**GPIO_SetBits(GPIOx, GPIO_Pin_x)** 就可以实现对**GPIOx**的**Pinx**位为高电平； **GPIO_ResetBits(GPIOx, GPIO_Pin_x)**就可以实现对**GPIOx**的**Pinx**位为低电平。

（3）具有外部中断/唤醒能力，端口配置成输入模式时，具有外部中断能力。

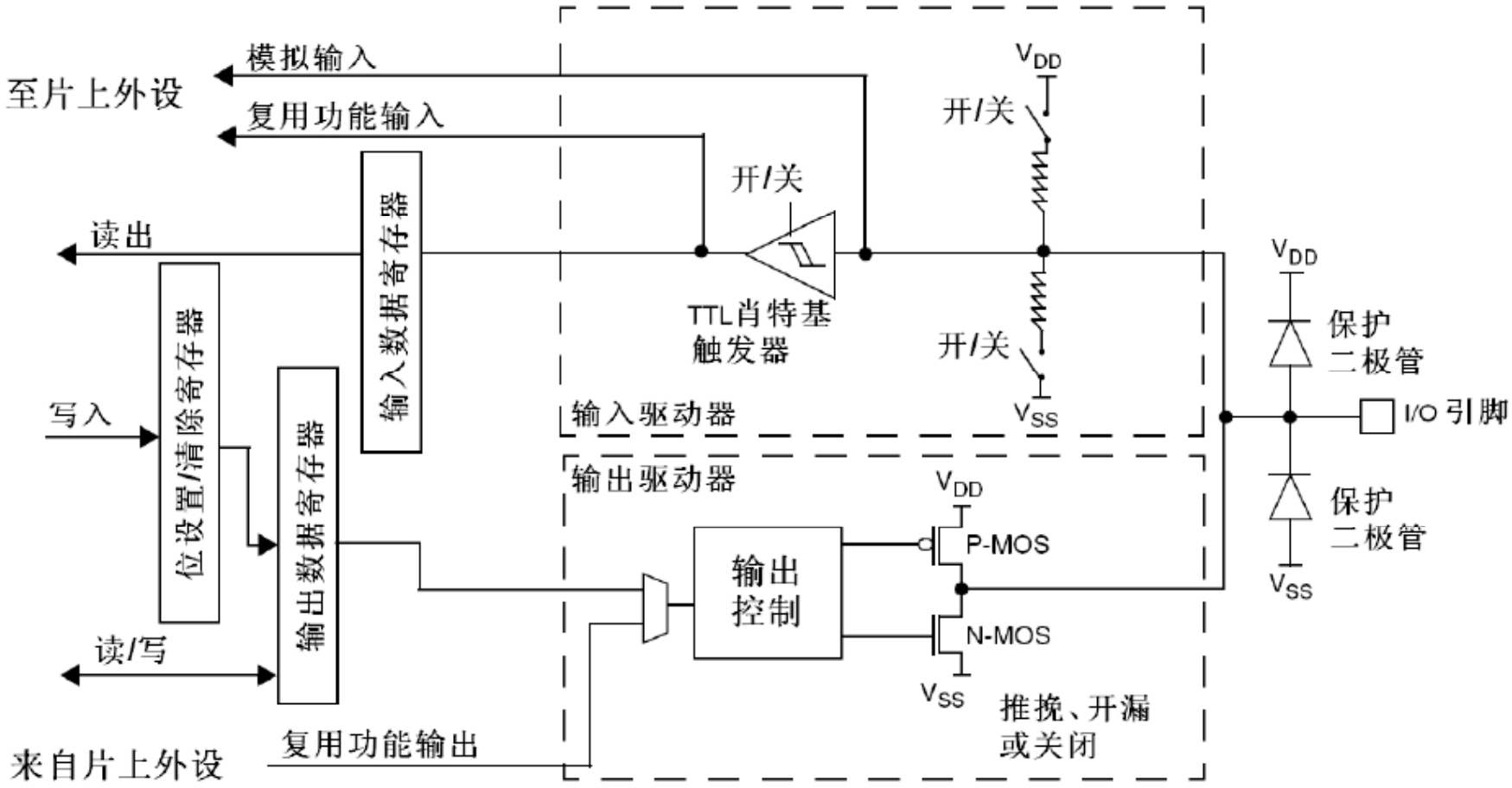
（4）具有复用功能，复用功能的端口兼有**I/O**功能等。

（5）软件重新映射**I/O**复用功能：为了使不同器件封装的外设**I/O**功能的数量达到最优，可以把一些服用功能重新映射到其他一些引脚上。这可以通过软件配置到相应的寄存器来完成。

（6）**GPIO**口的配置具有锁定机制。当配置好**GPIO**口后，在端口位上执行了锁定（**LOCK**），可以通过程序锁住配置组合，在下一次复位之前，将不能再更改端口位的配置。

一、 GPIO结构

I/O端口位的基本结构图

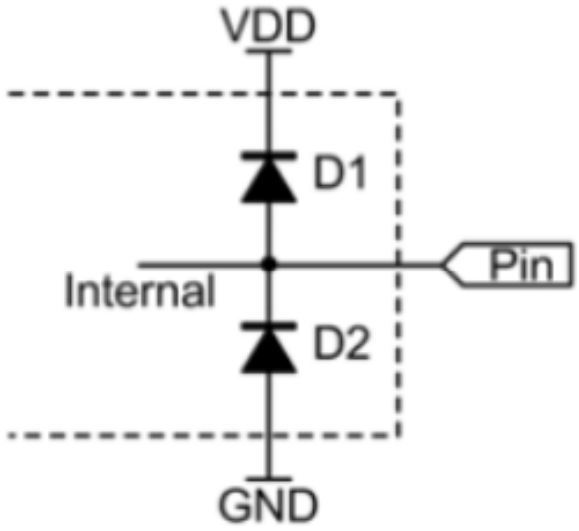


钳位保护二极管

GPIO内部具有钳位保护二极管，其作用是防止从外部管脚**Pin**输入的电压过高或者过低。

VDD正常供电是**3.3V**。如果从**Pin**输入的信号（假设任何输入信号都有一定的内阻）电压超过**VDD**加上二极管**D1**的导通压降（假定在**0.6V**左右），则二极管**D1**导通，会把多于的电流引到**VDD**，而真正输入到内部的信号电压不会超过**3.9V**。

同理，如果从**Pin**输入的信号电压比**GND**还低，则由于二极管**D2**的作用，会把实际输入内部的信号电压钳制在 **–0.6V**左右。



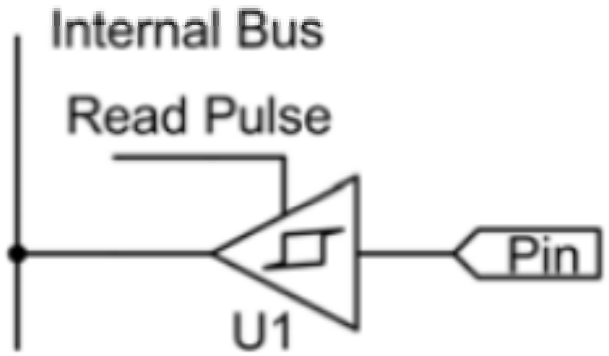
GPIO的配置种类

对于GPIO的配置种类有8种之多：

- （1）GPIO_Mode_AIN 模拟输入
- （2）GPIO_Mode_IN_FLOATING 浮空输入
- （3）GPIO_Mode_IPD 下拉输入
- （4）GPIO_Mode_IPU 上拉输入
- （5）GPIO_Mode_Out_OD 开漏输出
- （6）GPIO_Mode_Out_PP 推挽输出
- （7）GPIO_Mode_AF_OD 复用开漏输出
- （8）GPIO_Mode_AF_PP 复用推挽输出

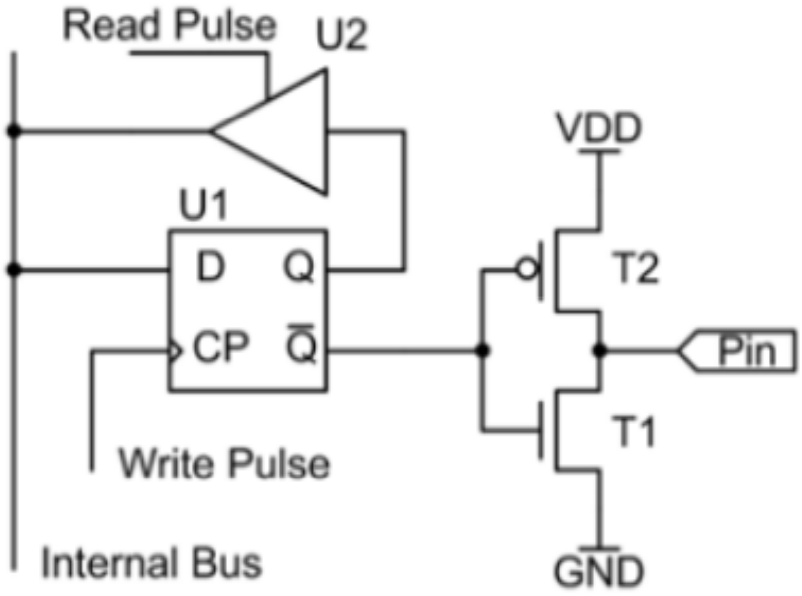
浮空输入（高阻输入）

- 输入模式的结构比较简单，就是一个带有施密特触发输入（**Schmitt-triggered input**）的三态缓冲器（**U1**），并具有很高的输入等效阻抗。
- 施密特触发输入的作用是能将缓慢变化的或者是畸变的输入脉冲信号整形成比较理想的矩形脉冲信号。
- 执行**GPIO**管脚读操作时，在读脉冲（**Read Pulse**）的作用下会把管脚（**Pin**）的当前电平状态读到内部总线上（**Internal Bus**）。
- 在不执行读操作时，外部管脚与内部总线之间是隔离的。

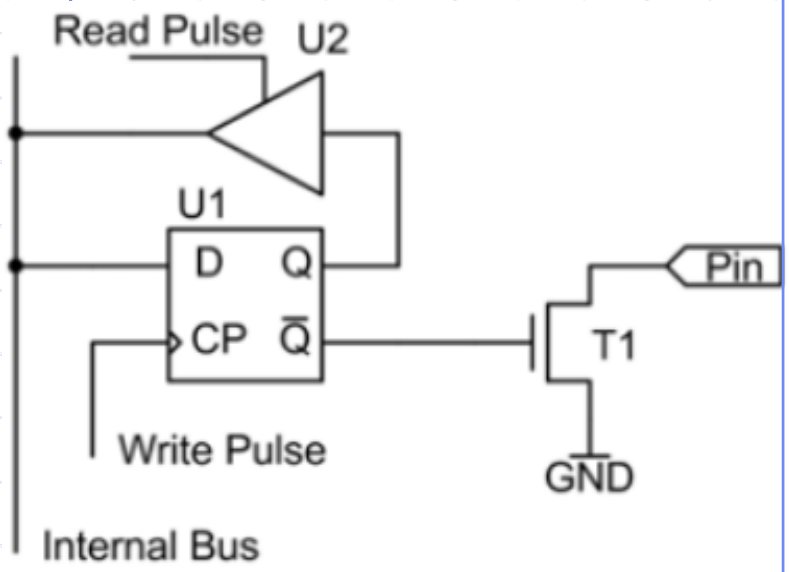


推挽输出

- 执行GPIO管脚写操作时，在写脉冲（Write Pulse）的作用下，数据被锁存到Q和/Q。
- T1和T2构成CMOS反相器，T1导通或T2导通时都表现出较低的阻抗，但T1和T2不会同时导通或同时关闭，最后形成的是推挽输出。



开漏输出



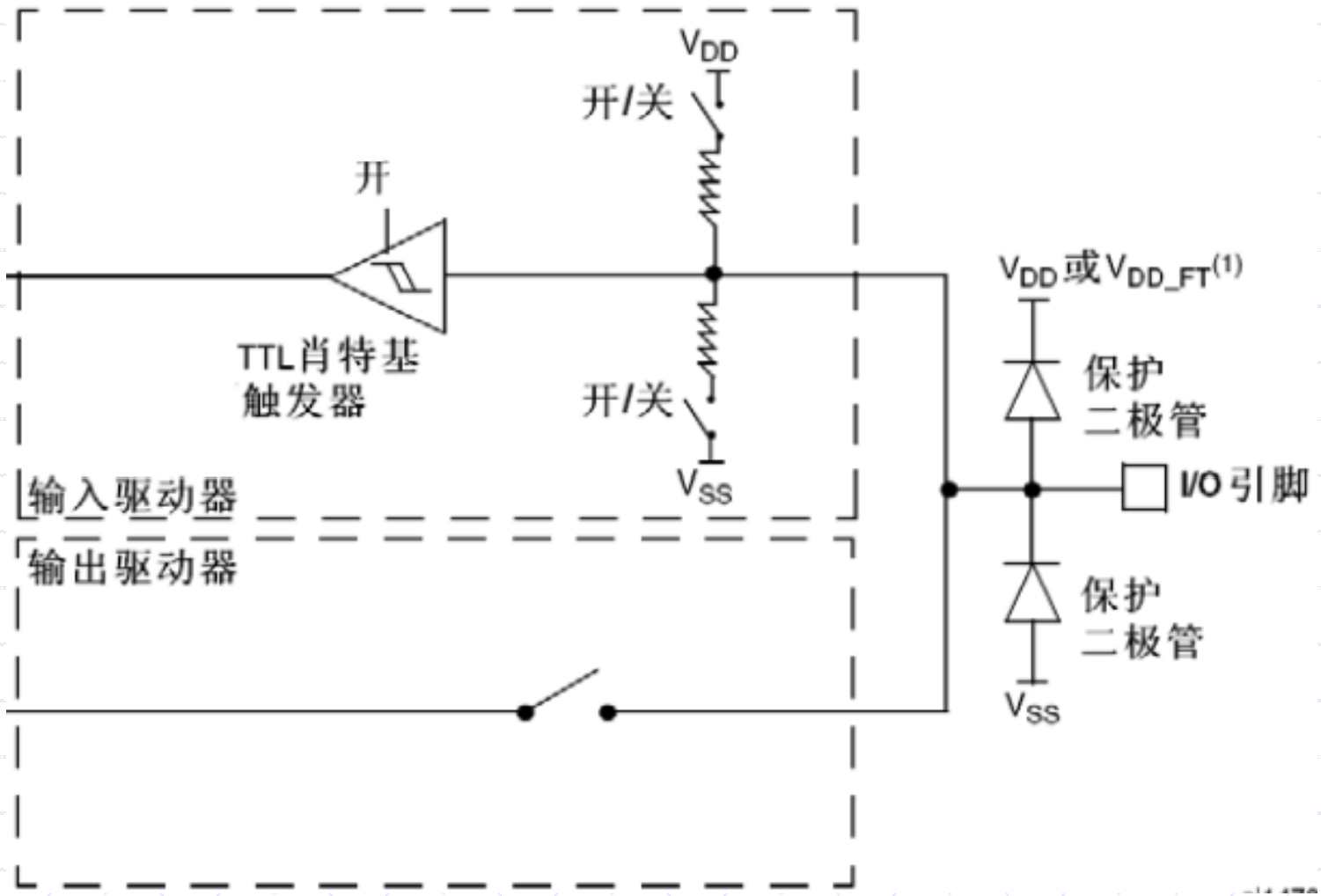
- 开漏输出只有下拉晶体管**T1**而没有上拉晶体管。同样，**T1**实际上也是多组可编程选择的晶体管。开漏输出的实际作用就是一个开关，输出“**1**”时断开、输出“**0**”时连接到**GND**。
- 开漏输出结构没有内部上拉，因此在实际应用时通常都要外接合适的上拉电阻（通常采用**4.7～10kΩ**）。
- 开漏输出能够方便地实现“线与”逻辑功能，即多个开漏的管脚可以直接并在一起使用，并统一外接一个合适的上拉电阻，就自然形成“逻辑与”关系。
- 开漏输出的另一种用途是能够方便地实现不同逻辑电平之间的转换（如**3.3V**到**5V**之间），只需外接一个上拉电阻，而不需要额外的转换电路。

输入配置

当I/O端口配置为输入时：

- 输出缓冲器被禁止
- 施密特触发输入被激活
- 根据输入配置(上拉，下拉或浮动)的不同，弱上拉和下拉电阻被连接
- 出现在I/O脚上的数据在每个APB2时钟被采样到输入数据寄存器
- 对输入数据寄存器的读访问可得到I/O状态

输入浮空/上拉/下拉配置图

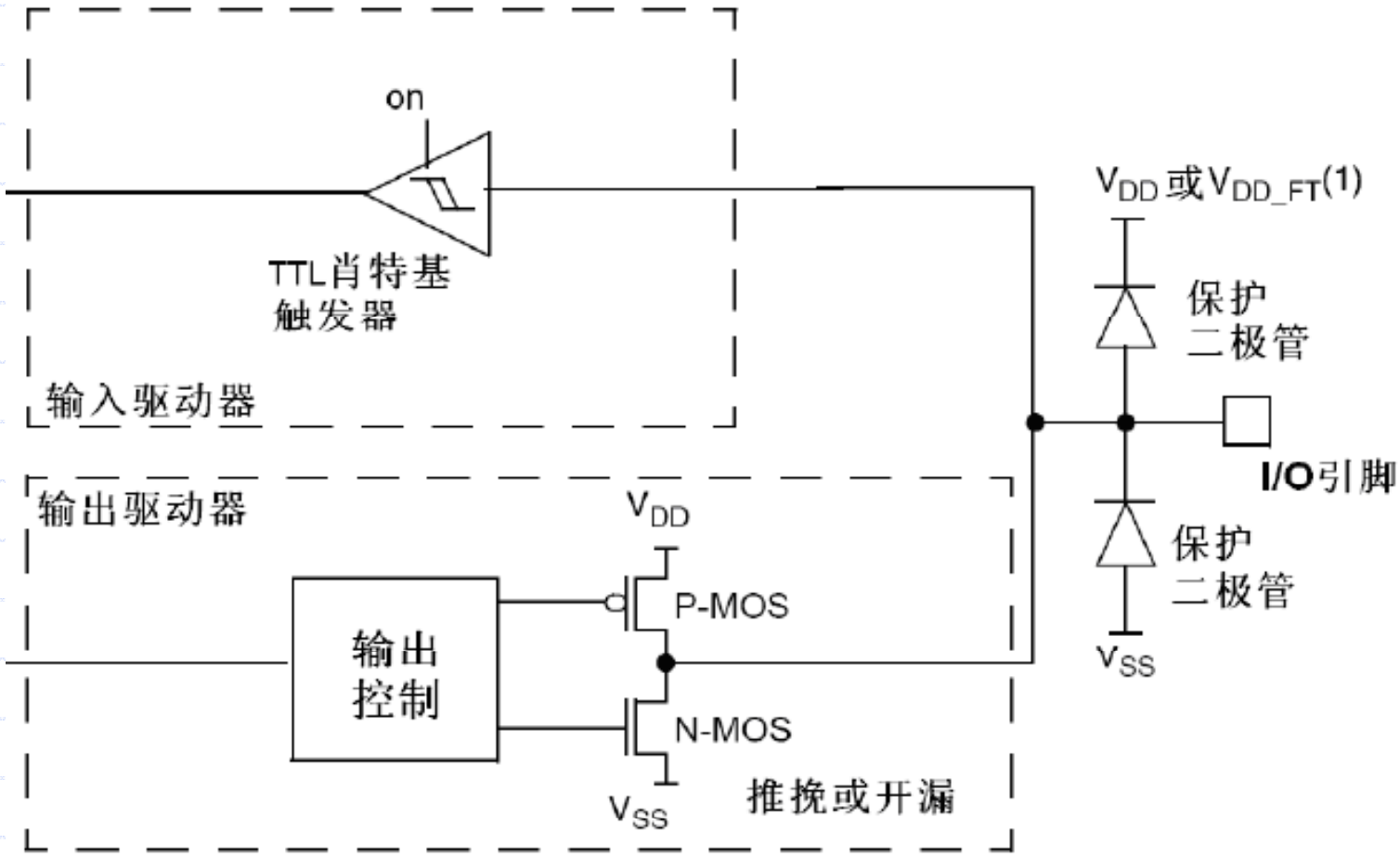


输出配置

当**I/O**端口被配置为输出时：

- 输出缓冲器被激活
  - 开漏模式：输出寄存器上的’**0**’激活**N-MOS**，而输出寄存器上的’**1**’将端口置于高阻状态(**P-MOS**从不被激活)。
  - 推挽模式：输出寄存器上的’**0**’激活**N-MOS**，而输出寄存器上的’**1**’将激活**P-MOS**。
- 施密特触发输入被激活
- 弱上拉和下拉电阻被禁止
- 出现在**I/O**脚上的数据在每个**APB2**时钟被采样到输入数据寄存器
- 在开漏模式时，对输入数据寄存器的读访问可得到**I/O**状态
- 在推挽模式时，对输出数据寄存器的读访问得到最后一次写的值。

I/O端口位的输出配置图



二、 GPIO寄存器

每个GPI/O端口有：

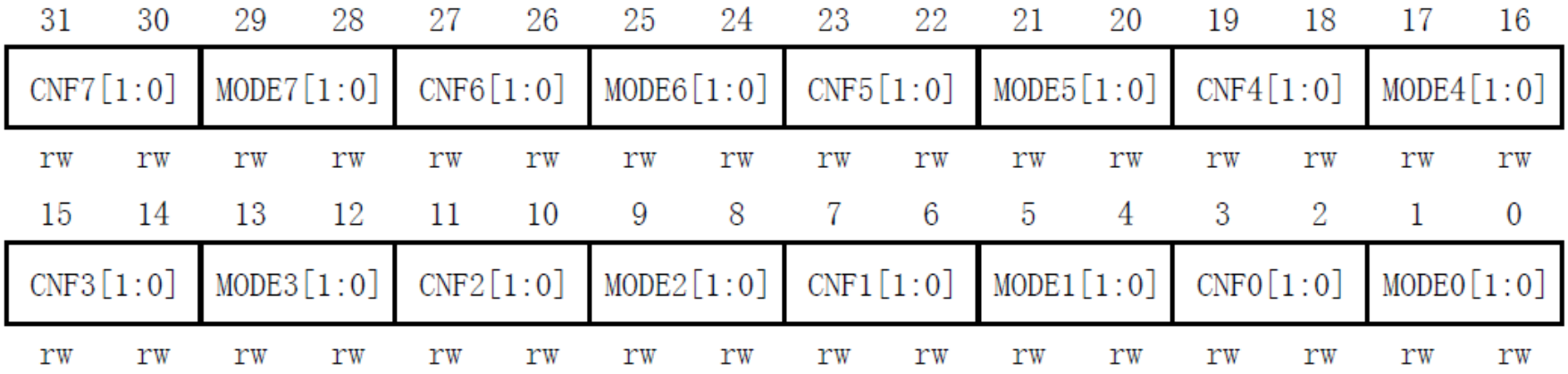
- 两个32位配置寄存器(GPIOx_CRL，GPIOx_CRH)，
- 两个32位数据寄存器(GPIOx_IDR，GPIOx_ODR)，
- 一个32位置位/复位寄存器(GPIOx_BSRR)，
- 一个16位复位寄存器(GPIOx_BRR)
- 一个32位锁定寄存器(GPIOx_LCKR)。

- GPIO端口的每个位可以由软件分别配置成多种模式。
- 必须以字(32位)的方式操作这些外设寄存器。
- GPIOx_BSRR和GPIOx_BRR寄存器允许对任何GPIO寄存器的读/更改的独立访问，这样，在读和更改访问之间产生IRQ时不会发生危险。

二、 GPIO寄存器

- **CRL** 端口配置低寄存器
- **CRH** 端口配置高寄存器
- **IDR** 端口输入数据寄存器
- **ODR** 端口输出数据寄存器
- **BSRR** 端口位设置/复位寄存器
- **BRR** 端口位复位寄存器
- **LCKR** 端口配置锁定寄存器
- **EVCR** 事件控制寄存器
- **MAPR** 复用重映射和调试
- **EXTICR** 外部中断线路**0-15**配置寄存器

端口配置低寄存器GPIOx_CRL) (x=A..E)
复位值：0x4444 4444



CNFy[1:0]：端口x配置位(y = 0...7) (Port x configuration bits)

MODEy[1:0]：端口x的模式位(y = 0...7) (Port x mode bits)

端口位配置表

- 复位期间和刚复位后，复用功能未开启，**I/O**端口被配置成浮空输入模式**(CNFx[1:0]=01b，MODEx[1:0]=00b)**。

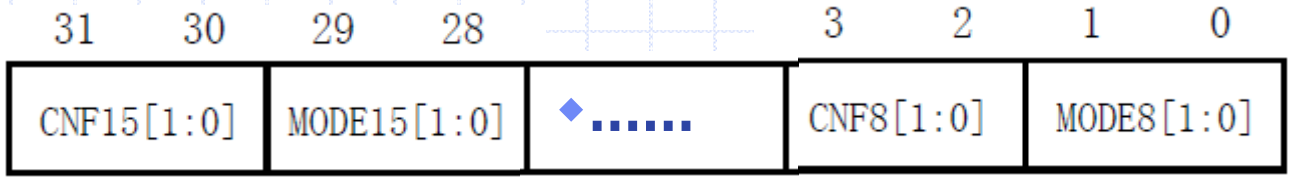
配置模式		CNF1	CNF0	MODE1	MODE0	PxODR寄存器
通用输出	推挽(Push-Pull)	0	0	01 10 11 见表18		0 或 1
	开漏(Open-Drain)		1			0 或 1
复用功能输出	推挽(Push-Pull)	1	0			不使用
	开漏(Open-Drain)		1			不使用
输入	模拟输入	0	0	00		不使用
	浮空输入		1			不使用
	下拉输入	1	0			0
	上拉输入					1

输出模式位

MODE[1:0]	意义
00	输入模式(复位后的状态)
01	最大输出速度为10MHz
10	最大输出速度为2MHz
11	最大输出速度为50MHz

端口配置高寄存器(GPIOx_CRH) (x=A..E)

复位值：**0x4444 4444**

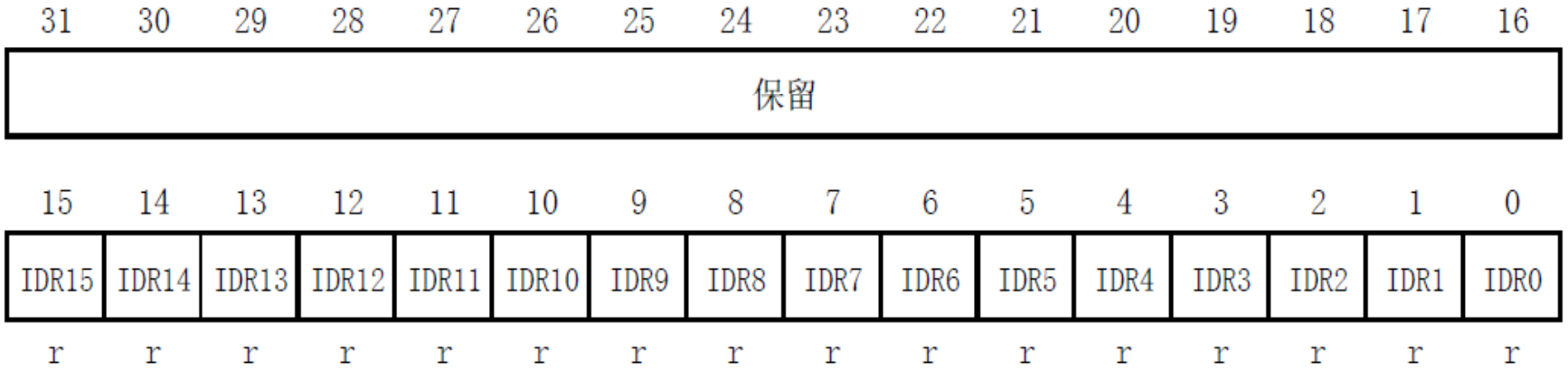


CNFy[1:0]：端口x配置位(y = 8...15) (Port x configuration bits)

MODEy[1:0]：端口x的模式位(y = 8...15) (Port x mode bits)

端口输入数据寄存器(GPIOx_IDR) (x=A..E)

复位值：0x0000 XXXX



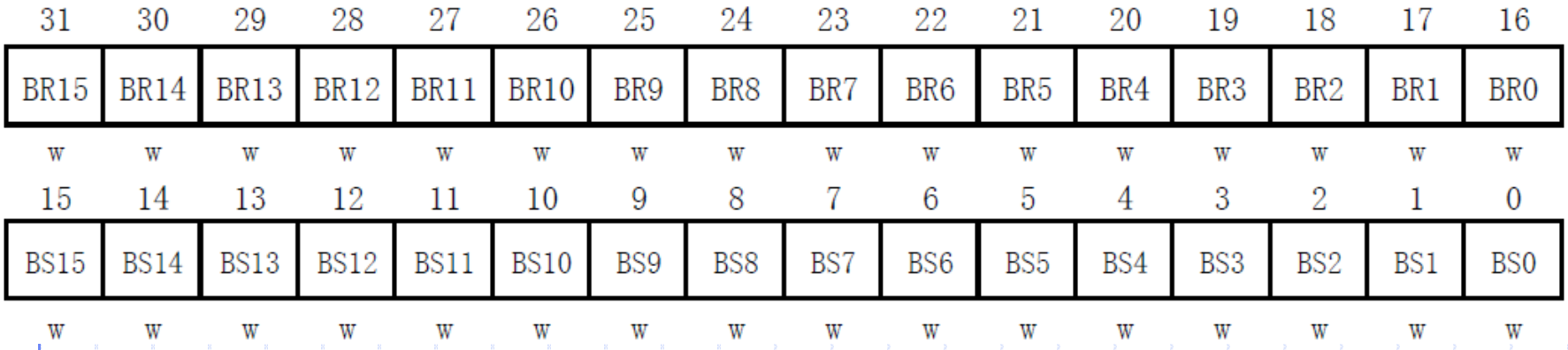
- IDRy[15:0]：端口输入数据(y = 0…15) (Port input data)
- 这些位为只读并只能以字(16位)的形式读出。
- 读出的值为对应I/O口的状态。

端口输出数据寄存器(GPIOx_ODR) (x=A..E)

复位值：**0x0000 0000**

- **ODRy[15:0]**：端口输出数据**(y = 0…15) (Port output data)**
- 这些位可读可写并只能以字**(16**位**)**的形式操作。
- 注：对**GPIOx_BSRR(x = A…E)**，可以分别地对各个**ODR**位进行独立的设置**/**清除。

端口位设置/清除寄存器(GPIOx_BSRR)
复位值：0x0000 0000



- BRy：清除端口x的位y（y = 0…15）（Port x Reset bit y）这些位只能写入并只能以字(16位)的形式操作。
- 0：对对应的ODRy位不产生影响，1：清除对应的ODRy位为0

- BSy：设置端口x的位y。
- 0：对对应的ODRy位不产生影响，1：设置对应的ODRy位为1

- 注：如果同时设置了BSy和BRy的对应位，BSy位起作用。

端口位清除寄存器(GPIOx_BRR)
复位值：0x0000 0000

- **BRy: 清除端口x的位y (y = 0...15) (Port x Reset bit y)**
- 这些位只能写入并只能以字(16位)的形式操作。
- **0**：对对应的**ODRy**位不产生影响，
- **1**：清除对应的**ODRy**位为**0**

端口配置锁定寄存器(GPIOx_LCKR)
复位值：0x0000 0000

- 当执行正确的写序列设置了位16(LCKK)时，该寄存器用来锁定端口位的配置。
- 位[15:0]用于锁定GPIO端口的配置。在规定的写入操作期间，不能改变LCKy[15:0]。当对相应的端口位执行了LOCK序列后，在下次系统复位之前将不能再更改端口位的配置。
- 每个锁定位锁定控制寄存器(CRL, CRH)中相应的4个位。
- **LCKy:** 端口x的锁位y (y = 0…15) (Port x Lock bit y) 。这些位可读可写但只能在LCKK位为0时写入。
- **0**：不锁定端口的配置， **1**：锁定端口的配置

端口配置锁定寄存器格式

GPIO寄存器结构

- **GPIO寄存器结构，*GPIO_TypeDef*和*AFIO_TypeDef*，在文件*“stm32f10x_map.h”*中定义如下：**

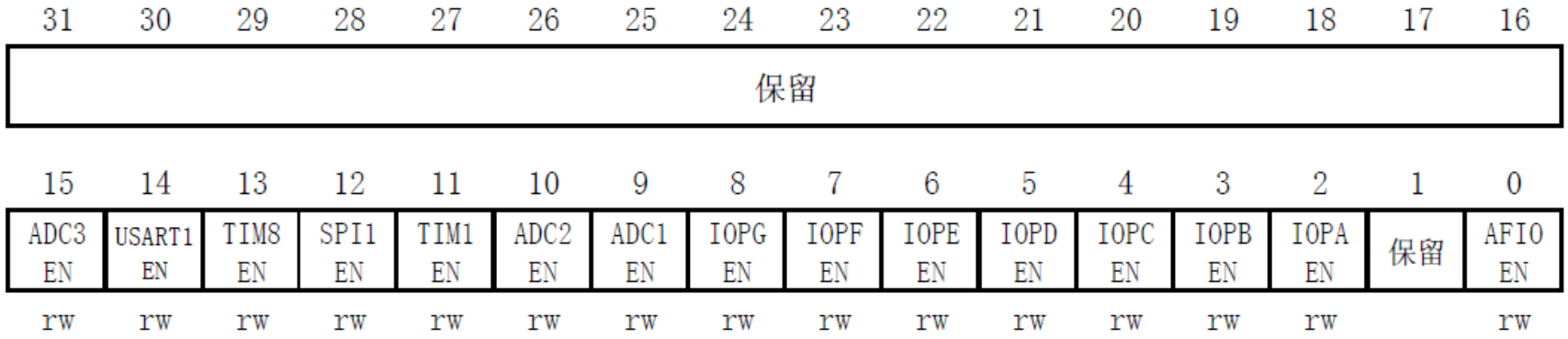
```
typedef struct
{
vu32 CRL;
vu32 CRH;
vu32 IDR;
vu32 ODR;
vu32 BSRR;
vu32 BRR;
vu32 LCKR;
} GPIO_TypeDef;
```

```
typedef struct
{
vu32 EVCR;
vu32 MAPR;
vu32 EXTICR[4];
} AFIO_TypeDef;
```

寄存器编程举例

```
void LED_Init(void)
{
    RCC->APB2ENR|=1<<6;        //使能PORTE时钟
    GPIOE->CRL&=0X00000000;
    GPIOE->CRL|=0X33333333;  //PE.0-PE.7推挽输出
    GPIOE->ODR|=0xFF;          //PE.0-PE.7输出高电平
}
```

APB2外设时钟使能寄存器(RCC_APB2ENR)
复位值：0x0000 0000



- **IOPEEN(位6)：IO端口E时钟使能 (I/O port E clock enable)** 由软件置’1’或清’0’
- **0：IO端口E时钟关闭； 1：IO端口E时钟开启。**

例：灯的控制

```
//点亮灯
GPIOC->ODR = 0xfffffffe;        //点亮1个灯
GPIOC->BSRR = 0x00030000;//点亮2个灯
GPIOC->BRR = 0x000C;
//GPIO_ResetBits(GPIOC, GPIO_Pin_2);
//GPIO_Write(GPIOC, 0xffff);
//GPIO_WriteBit(GPIOC, GPIO_Pin_3, Bit_RESET);
//熄灭灯
GPIOC->ODR = 0xffffffff;
GPIOC->BSRR = 0x00000003;
GPIOC->BSRR = 0x0000000C;
//GPIO_SetBits(GPIOC, GPIO_Pin_2);
//GPIO_Write(GPIOC, 0xfff0);
//GPIO_WriteBit(GPIOC, GPIO_Pin_3, Bit_SET);
```

三、 GPIO函数

- **GPIO_DeInit** 将外设**GPIOx**寄存器重设为缺省值
- **GPIO_AFIODeInit**

  将复用功能（重映射事件控制和**EXTI**设置）重设为缺省值
- **GPIO_Init**

  根据**GPIO_InitStruct**中指定的参数初始化外设**GPIOx**寄存器
- **GPIO_StructInit**

  把**GPIO_InitStruct**中的每一个参数按缺省值填入
- **GPIO_ReadInputDataBit** 读取指定端口管脚的输入
- **GPIO_ReadInputData** 读取指定的**GPIO**端口输入
- **GPIO_ReadOutputDataBit** 读取指定端口管脚的输出
- **GPIO_ReadOutputData** 读取指定的**GPIO**端口输出

三、 GPIO函数

- **GPIO_SetBits** 设置指定的数据端口位
- **GPIO_ResetBits** 清除指定的数据端口位
- **GPIO_WriteBit** 设置或者清除指定的数据端口位
- **GPIO_Write** 向指定**GPIO**数据端口写入数据
- **GPIO_PinLockConfig** 锁定**GPIO**管脚设置寄存器
- **GPIO_EventOutputConfig** 选择**GPIO**管脚用作事件输出
- **GPIO_EventOutputCmd** 使能或者失能事件输出
- **GPIO_PinRemapConfig** 改变指定管脚的映射
- **GPIO_EXTILineConfig** 选择**GPIO**管脚用作外部中断线路

函数GPIO_DeInit

- 功能：将外设GPIOx寄存器重设为缺省值。
- 函数原形：void GPIO_DeInit(GPIO_TypeDef* GPIOx)
- GPIOx：x可以是A，B，C，D或者E，来选择GPIO外设。
- 被调用函数： RCC_APB2PeriphResetCmd()
- 例：

```
/* Resets the GPIOA peripheral registers to their default reset values */
GPIO_DeInit(GPIOA);
```

函数GPIO_AFIODeInit

- 功能描述：将复用功能（重映射事件控制和**EXTI**设置）重设为缺省值
- 函数原形：**void GPIO_AFIODeInit(void)**
- 被调用函数：**RCC_APB2PeriphResetCmd()**
- 例：

```
/* Resets the Alternate functions registers to their default reset values */
GPIO_AFIODeInit();
```

函数GPIO_Init

- 功能描述：根据**GPIO_InitStruct**中指定的参数初始化外设**GPIOx**寄存器
- 函数原形：

  void GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
- **GPIOx**：**x**可以是**A**，**B**，**C**，**D**或者**E**，来选择**GPIO**外设
- **GPIO_InitStruct**：指向结构**GPIO_InitTypeDef**的指针，包含了外设**GPIO**的配置信息
- 例如：**GPIO_Init (GPIOA, &GPIO_InitStructure);**

- **GPIO_InitTypeDef定义于文件 *“stm32f10x_gpio.h”*:**

```
typedef  struct
    {
     u16  GPIO_Pin;
     GPIOSpeed_TypeDef  GPIO_Speed;
     GPIOMode_TypeDef  GPIO_Mode;
    } GPIO_InitTypeDef;
```

GPIO_Pin值

该参数选择待设置的**GPIO**管脚，使用操作符 “|”可以一次选中多个管脚。可以使用下表中的任意组合。

- **GPIO_Pin_None:** 无管脚被选中
- **GPIO_Pin_0:** 选中管脚**0**
- **GPIO_Pin_1:** 选中管脚**1**
- **GPIO_Pin_2:** 选中管脚**2**
- **GPIO_Pin_3:** 选中管脚**3**
- **……**
- **GPIO_Pin_All:** 选中全部管脚

GPIO_Speed值

GPIO_Speed_10MHz	最高输出速率 10MHz
GPIO_Speed_2MHz	最高输出速率 2MHz
GPIO_Speed_50MHz	最高输出速率 50MHz

GPIO_Mode: 用以设置选中管脚的工作状态

GPIO_Mode	描述
GPIO_Mode_AIN	模拟输入
GPIO_Mode_IN_FLOATING	浮空输入
GPIO_Mode_IPD	下拉输入
GPIO_Mode_IPU	上拉输入
GPIO_Mode_Out_OD	开漏输出
GPIO_Mode_Out_PP	推挽输出
GPIO_Mode_AF_OD	复用开漏输出
GPIO_Mode_AF_PP	复用推挽输出

GPIO的初始化配置程序

例：

```
/* Configure all the GPIOA in Input Floating mode */
GPIO_InitTypeDef  GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
GPIO_InitStructure.GPIO_Speed =
   GPIO_Speed_10MHz;
GPIO_InitStructure.GPIO_Mode =
   GPIO_Mode_IN_FLOATING;
GPIO_Init (GPIOA, &GPIO_InitStructure);
```

函数GPIO_StructInit

函数名	GPIO_StructInit
函数原形	void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	把 GPIO_InitStruct 中的每一个参数按缺省值填入
输入参数	GPIO_InitStruct：指向结构 GPIO_InitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

成员	缺省值
GPIO_Pin	GPIO_Pin_All
GPIO_Speed	GPIO_Speed_2MHz
GPIO_Mode	GPIO_Mode_IN_FLOATING

例： **/* Initialize the GPIO Init Structure parameters */**

```
GPIO_InitTypeDef  GPIO_InitStructure;
GPIO_StructInit (&GPIO_InitStructure);
```

函数 GPIO_ReadInputDataBit

- 函数名：**GPIO_ReadInputDataBit**
- 函数原形：

  u8 GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
- 功能描述： 读取指定端口管脚的输入
- 输入参数**1**：

  GPIOx----x可以是**A**，**B**，**C**，**D**或者**E**，来选择**GPIO**外设
- 输入参数**2**： **GPIO_Pin----**待读取的端口位
- 返回值：输入端口管脚值

例：

```
/* Reads the seventh pin of the GPIOB and
   store it in ReadValue variable */
u8 ReadValue;
ReadValue = GPIO_ReadInputDataBit (GPIOB, GPIO_Pin_7);
```

函数GPIO_ReadInputData

- 函数名: **GPIO_ReadInputData**
- 函数原形: **u16 GPIO_ReadInputData(GPIO_TypeDef* GPIOx)**
- 功能描述: 读取指定的**GPIO**端口输入
- 输入参数: **GPIOx**：**x**可以是**A**，**B**，**C**，**D**或者**E**，来选择**GPIO**外设
- 输出参数: 无
- 返回值: **GPIO**输入数据端口值

例:

```
/*Read the GPIOC input data port and store it
  in ReadValue variable*/
u16 ReadValue;
ReadValue = GPIO_ReadInputData (GPIOC);
```

函数GPIO_ReadOutputDataBit

- 函数名**GPIO_ReadOutputDataBit**
- 函数原形:

  u8 GPIO_ReadOutputDataBit (GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
- 功能描述 读取指定端口管脚的输出

  输入参数**1----GPIOx**：**x**可以是**A**，**B**，**C**，**D**或者**E**，来选择**GPIO**外设

  输入参数**2----GPIO_Pin**：待读取的端口位
- 返回值: 输出端口管脚值

例：
```
/* Reads the seventh pin of the GPIOB and store it
   in ReadValue variable */
u8 ReadValue;
ReadValue = GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_7);
```

函数GPIO_ReadOutputData

- 函数名: **GPIO_ReadOutputData**
- 函数原形:

  u16 GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)
- 功能描述: 读取指定的**GPIO**端口输出
- 输入参数: **GPIOx----x**可以是**A**，**B**，**C**，**D**或者**E**，来选择**GPIO**外设
- 返回值: **GPIO**输出数据端口值

例：
```
/* Read the GPIOC output data port and store it
   in ReadValue variable */
u16 ReadValue;
ReadValue = GPIO_ReadOutputData(GPIOC);
```

函数GPIO_SetBits

- 函数名**GPIO_SetBits**
- 函数原形

  void GPIO_SetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
- 功能描述 设置指定的数据端口位
- 输入参数**1 GPIOx**：**x**可以是**A**，**B**，**C**，**D**或者**E**，来选择**GPIO**外设
- 输入参数**2 GPIO_Pin**：待设置的端口位该参数可以取**GPIO_Pin_x(x**可以是**0-15)**的任意组合

例： **/* Set the GPIOA port pin 10 and pin 15 */**

GPIO_SetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);

函数GPIO_ResetBits

- 函数原形:
- **void GPIO_ResetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)**
- 功能描述: 清除指定的数据端口位

例： **/* Clears the GPIOA port pin 10 and pin 15 */**

```
GPIO_ResetBits(GPIOA, GPIO_Pin_10 |
  GPIO_Pin_15)
```

函数GPIO_WriteBit

- 函数原形:

  void GPIO_WriteBit (GPIO_TypeDef* GPIOx, u16 GPIO_Pin, BitAction BitVal)

- 功能描述: 设置或者清除指定的数据端口位
- **BitVal:** 该参数指定了待写入的值. 该参数必须取枚举**BitAction**的其中一个值. **Bit_RESET:** 清除数据端口位, **Bit_SET:** 设置数据端口位

例：

```
/* Set the GPIOA port pin 15 */
    GPIO_WriteBit(GPIOA, GPIO_Pin_15, Bit_SET);
```

函数GPIO_Write

- 函数原形:

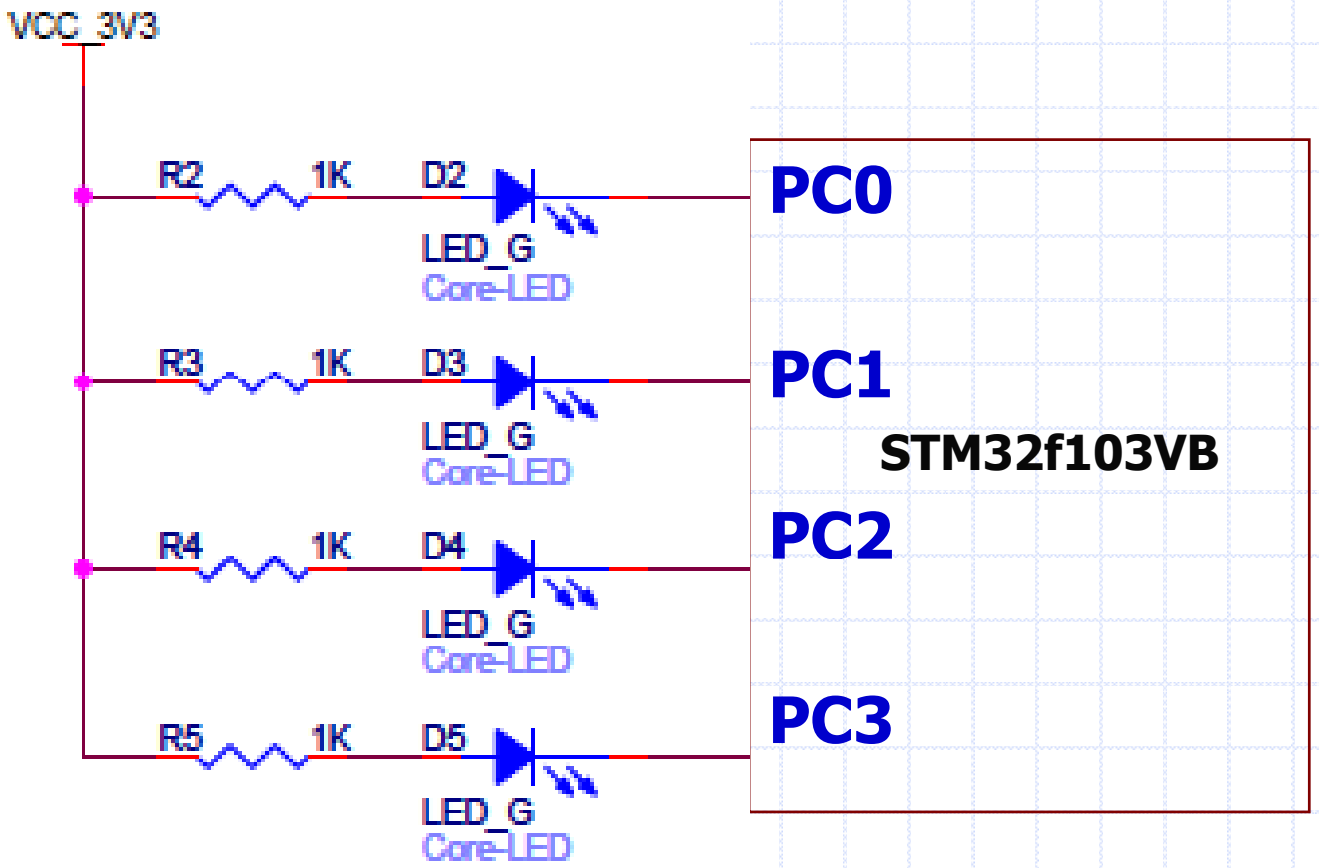
  void GPIO_Write(GPIO_TypeDef* GPIOx, u16 PortVal)

- 功能描述: 向指定**GPIO**数据端口写入数据
- **PortVal:** 待写入端口数据寄存器的值

例： **/* Write data to GPIOA data port */**

GPIO_Write(GPIOA, 0x1101);

四、 GPIO编程举例---点亮一个灯



主程序

```
#include "stm32f10x.h"
ErrorStatus HSEStartUpStatus;  //定义外部高速晶振启动状态枚举变量
void Delay(vu32 nTime);          //vu32代表无符号32位长整型
void RCC_Configuration(void);
void GPIO_Configuration(void);
int main(void)
{
    RCC_Configuration();              //配置时钟
    GPIO_Configuration();             //配置端口
    for( ; ; )                        //或者  while(1)
   {
        GPIOC->ODR = 0xfffffffe;
         Delay(800000);
       GPIOC->ODR = 0xffffffff;
         Delay(800000);
   }
}
```

```
void RCC_Configuration(void)          //配置时钟的函数
{
    RCC_DeInit();
    RCC_HSEConfig(RCC_HSE_ON);
    HSEStartUpStatus = RCC_WaitForHSEStartUp();
 if (HSEStartUpStatus == SUCCESS)
 {
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    RCC_PCLK2Config(RCC_HCLK_Div1);
    RCC_PCLK1Config(RCC_HCLK_Div2);
    FLASH_SetLatency(FLASH_Latency_2);
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
    RCC_PLLCmd(ENABLE);
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
    while(RCC_GetSYSCLKSource() != 0x08);
 }
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
}
```

配置端口函数

```
void GPIO_Configuration(void)
{   //端口GPIOC[3:0]，推挽输出，50MHz。
    GPIO_InitTypeDef GPIO_InitStructure;
     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1
    |  GPIO_Pin_2 | GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
     //端口GPIOA[3:0]，浮空输入，2MHz（可省略）。
     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1
    | GPIO_Pin_2 | GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode =
    GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

延时函数

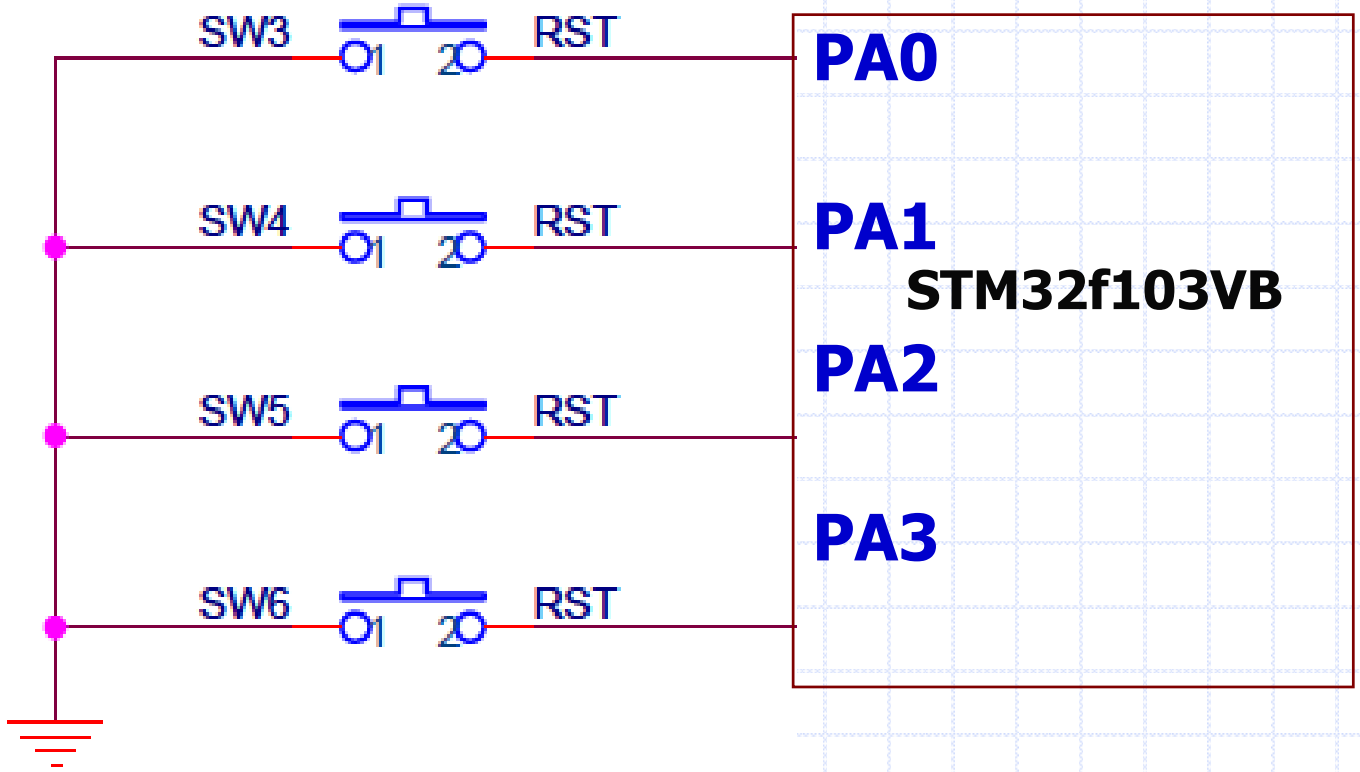
```
void Delay(vu32 nCount)
{
    for( ; nCount != 0; nCount--);
}
```

关于ErrorStatus

```
typedef enum
{
        ERROR = 0,
        SUCCESS = !ERROR
} ErrorStatus;
```

GPIO编程举例---独立按键输入

- 核心板上，按键接口电路：



```
#include  "stm32f10x.h"
ErrorStatus HSEStartUpStatus;
void Delay(vu32 nTime);
void RCC_Configuration(void);
void GPIO_Configuration(void);
u8   t;
```

主函数

```
int main (void)
{
    RCC_Configuration();          /* 配置系统时钟，使能外设时钟 */
    GPIO_Configuration();          /* 配置GPIO，PC推挽输出，PA浮空输入
    */
    GPIOC->ODR = 0xffffffff;       /* 灯全灭 */
    while(1)
    {
          t = GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_3);
          if (t==0)
          {
             GPIOC->ODR = 0xfffffff0;
          }
          t = GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_2);
          if (t==0)
          {
             GPIOC->ODR = 0xfffffff3;
    }
}
```

```
//配置系统时钟
void RCC_Configuration(void)
{
 // 使能外设（PA口和PC口）时钟
 RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOA
   | RCC_APB2Periph_GPIOC, ENABLE);
}
```

```
void GPIO_Configuration(void)
{   //配置GPIO, 推挽输出，上拉输入
    GPIO_InitTypeDef  GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 |
     GPIO_Pin_2 | GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 |
          GPIO_Pin_2 | GPIO_Pin_3;
     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
     GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

练习题

1. GPIO的英文全称是（）。
2. GPIO引脚通常分为组为（）、（）、（）、（）、（）。
3. 每组GPIO寄存器中每位对应的位置分别编号为（）--（）。
4. GPIO输入/输出模式有（）种。
5. GPIO常用工作模式有3种：（）、（）、（）。
6. STM32复位之后，所有端口被设置成（）方式。
7. 开漏输出时，端口只能输出（）电平，不能输出（）电平。
8. 用来设置工作模式的寄存器是（）、（）。
9. I/O寄存器必须以（）形式访问。
10. 端口数据输出寄存器是（）。
11. GPIOx_BSRR是（）位的寄存器。
12. 通过对GPIOx_BSRR寄存器的相应位写（），可以实现置位/复位。
13. GPIO有3种输出速度可选分别为（）、（）、（）。这里速度是指I/O口驱动电路的响应速度而不是输出信号的速度。
14. GPIO库函数屏蔽了对（）的操作，直接通过参数设置实现相应初始化、读写端口等功能。
15. 设置一位的函数有（）、（）、（）。
16. 同时写多位端口的函数是（）。

谢谢！